VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Aplikace teorie kódování

Application of Coding Theory

2020 Jakub Salamon

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Zadání diplomové práce

Student:	**Bc. Jakub Salamon**
Studijní program:	N2647 Informační a komunikační technologie
Studijní obor:	1103T031 Výpočetní matematika
Téma:	Aplikace teorie kódování Applications of Coding Theory
Jazyk vypracování:	čeština

Zásady pro vypracování:

Diplomová práce má za úkol s využitím stavebnice například LEGO Mindstroms demonstrovat na praktických příkladech využití samoopravných kódů. Stavebnice LEGO Mindstroms a podobné umožňují sestavovat zařízení, která s využitím signálních prvků a čidel spolu komunikují. Tato komunikace není vždy 100% a tato diplomová práce by měla
1) popsat a rozebrat vhodné samoopravné kódy s různou úrovní oprav chyb,
2) navrhnout uspořádání experimentu s přenosem v zašuměném kanálu,
3) demonstrovat využití samoopravných kódů při zajištění komunikace i v zašuměném kanálu,
4) zhodnotit navržené kódy i vhodnost uspořádání experimentu.
Je možné, že se ukáže, že stavebnice není nejvhodnějším prostředím pro praktickou demonstraci, což by mělo být v práci řádně zdůvodněno a neznamená to neúspěch diplomové práce.

Seznam doporučené odborné literatury:

R. Hill: A first Course in Coding Theory, Oxford Press, 2009

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Tereza Kovářová, Ph.D.**

Datum zadání: 01.09.2019
Datum odevzdání: 30.04.2020

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

prof. RNDr. Jiří Bouchala, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2020 ....Salamon.....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 30. dubna 2020 ..... Salamon .....

Na tomto místě bych rád poděkoval vedoucí práce paní Mgr. Tereze Kovářové, Ph.D. za ochotu, pomoc, cenné rady a připomínky. Dále panu doc. Mgr. Petru Kovářovi, Ph.D. za vymyšlení tématu této práce a jeho postřehy a komentáře. V neposlední řadě bych rád poděkoval také své manželce a rodině za podporu.

Abstrakt

Teorie kódování se zabývá reprezentací informace za účelem získání jistých vlastností. V této diplomové práci nás bude zajímat schopnost opravování chyb vzniklých při přenosu informace s využitím samoopravných kódů. Hlavním předmětem je experiment, který má za úkol vysvětlit laikům princip fungování samoopravných kódů populárně naučnou formou. Z toho důvodu je k jeho realizaci použita programovatelná stavebnice Lego Mindstorms. Kromě popisu provedení, funkčnosti a rozboru zdrojových kódů je uveden i potřebný matematický aparát, který slouží nejen pro manipulaci se samoopravnými kódy, ale také k jejich srovnání, aby bylo možno zvolit si optimální kód pro danou situaci. Vzorem pro experiment se stala americká vesmírná sonda Juno, která je v práci taktéž rozebrána.

Klíčová slova: teorie kódování, samoopravné kódy, blokové kódy, lineární kódy, kódování, dekódování, spolehlivost přenosu, Juno, Lego Mindstorms, popularizace

Abstract

Coding theory handles representation of information for a purpose of obtaining certain properties. In this thesis we are going to focus on correction of errors that occurred during information transfer using error-correcting codes. The main subject of this thesis is an experiment, aim of which is to demonstrate and clarify the basic principles of error correction to lay public in a popular educational way. For that reason, the experiment is realized with programmable building kit Lego Mindstorms. This thesis comprises not only a description of the experiment execution, functionality and source codes but also necessary mathematical apparatus. Using this apparatus we can work with error-correcting codes and also compare them so that we may choose optimal codes for given situations. The experiment was inspired by an American space probe called Juno, which is also described in the thesis.

Keywords: coding theory, error-correcting codes, block codes, linear codes, coding, decoding, data transfer reliability, Juno, Lego Mindstorms, popularisation

Obsah

Seznam použitých zkratek a symbolů

GF	–	Galois Field
BCH	–	Bose-Chaudhuri-Hocquenghem
RS	–	Reed-Solomonův
MDS	–	Maximum Distance Separable
CD	–	Compact Disc
DVD	–	Digital Versatile Disc
QR	–	Quick Response
LTE	–	Long Term Evolution

Seznam obrázků

Seznam tabulek

1 Úvod

Teorie kódování se zabývá reprezentací informace za účelem získání jistých vlastností. Zahrnuje tři hlavní odvětví. Prvním je komprese dat, kde nás zajímá, jak informace efektivně uchovávat. Druhým je kryptografie, jejímž hlavním objektem zájmu je ochrana informace před nepovolanými osobami. V neposlední řadě pak teorie kódování zahrnuje také ochranu dat proti chybám, způsobeným vnějšími vlivy, užitím tzv. samoopravných kódů. V této práci se budeme zabývat pouze posledním zmíněným odvětvím.

Ačkoliv se setkáme se samoopravnými kódy prakticky všude, kde se přenášejí informace z jednoho místa na druhé, ne všichni o jejich existenci vědí a ještě méně z nich rozumí, na jakém principu fungují. Přitom základní myšlenka je velice jednoduchá. Upravíme informaci tak, abychom byli schopni poznat, že se změnila a v nejlepším případě si i domyslet, jak se změnila. K tomuto úkolu se využívá matematický aparát, který je někdy velice jednoduchý, jindy poměrně složitý. Vesměs záleží na tom, jak moc efektivní chceme při kódování a dekódování být.

Tato diplomová práce je úzce spjatá s předchozí bakalářskou prací [13] a některé informace se z povahy zadání a ucelenosti textu musejí zopakovat. Většina opakujících se informací je ovšem doplněna a rozšířena o nové. Nejprve v kapitole 2.1 představíme pojmy “kód”, “samoopravný kód” a vysvětlíme základní princip detekce a opravování chyb. Následně jsou představeny atributy, pomocí kterých můžeme kódy definovat a srovnávat, abychom byli schopni vybrat si a sestrojit optimální kód pro danou situaci. Detailně je pak v kapitole 2.3 rozebrána skupina tzv. lineárních kódů, pro které mimo jiné počítáme spolehlivost přenosu dat s použitím právě těchto kódů. Dále se v kapitole 3 věnujeme konkrétní reálné aplikaci, ve které se můžeme se samoopravnými kódy setkat, americké vesmírné sondě Juno. Juno se stala jakýmsi předobrazem (vzorem) pro návrh a realizaci experimentu, kterému je věnován zbytek práce. V kapitole 4 pro experiment konstruujeme optimální samoopravný kód. Kapitola 5 pak rozebírá funkcionalitu a přiložené zdrojové kódy.

Právě experiment je hlavním cílem této práce a veškerá teorie a matematický aparát jsou uvedeny, aby bylo možno plně porozumět principu jeho fungování. Smyslem experimentu je představit samoopravné kódy široké veřejnosti ve srozumitelné, zábavné a co možná nejjednodušší formě a tím také popularizovat matematiku jako takovou. Jedná se o model přenosu informací zašuměným komunikačním kanálem realizovaný programovatelnou stavebnicí Lego Mindstorms.

2 Základní pojmy

V této kapitole zavedeme základní pojmy, které jsou nezbytné pro experiment. Jedná se o stručné shrnutí všech důležitých informací tak, aby nebylo nutné při návrhu a zkoumání kódů pro experiment odkazovat na externí zdroje. Hlavním zdrojem informací pro tuto část byly anglické učebnice kurzů teorie kódování a abstraktní algebry [1–5]. Mnoho informací je také uvedeno v bakalářské práci [13], kam se odkazujeme především na důkazy některých základních vlastností kódů.

2.1 Samoopravný kód

Nejprve je nutné si specifikovat, co rozumíme pojmem "kód", "samoopravný kód" a jakými vlastnostmi můžeme jednotlivé kódy popisovat a srovnávat.

Obecný kód C definujeme jako množinu posloupností (kódových slov) tvořených znaky zvolené abecedy. Abecedu budeme značit F_q, kde q představuje počet symbolů abecedy.

Můžeme si všimnout, že obecná definice kódu je velice benevolentní a neklade na kód v podstatě žádné požadavky.

Příklad 1
Pokud si jako abecedu zvolíme $F_{10} = \{0, 1, ..., 9\}$, kódem můžeme klidně nazvat množinu $C = \{123, 3134, 96, 815\}$. Pokud abecedou F_2 chápeme "čárku" a "tečku", můžeme sestavit Morseův kód. Případně pokud jako abecedu F_4 máme nukleové báze Adenin, Guanin, Cytosin a Thymin, můžeme jejich kombinací sestavit genetický kód. ■

Na kódová slova je často užitečné (má-li to smysl) pohlížet jako na aritmetické vektory. Pokud budeme mít například kódové slovo 123, můžeme jej podle potřeby chápat jako vektor $\vec{c} = (1, 2, 3)$.

Samoopravnými kódy budeme rozumět všechny kódy, které jsou schopné detekovat a opravovat chyby, které nastanou při přenosu dat chybovým komunikačním kanálem.

Příklad 2
Komunikačním kanálem byla odeslána zpráva 000 a vlivem vnějšího rušení došlo k záměně jednoho ze symbolů a přijata byla zpráva 001. Případně se mohl jeden symbol úplně ztratit a přijata byla zpráva 00. Kódy, které jsou schopny zprávu navrátit do původní podoby bez nutnosti zprávu odeslat znovu, nazveme samoopravnými kódy. ■

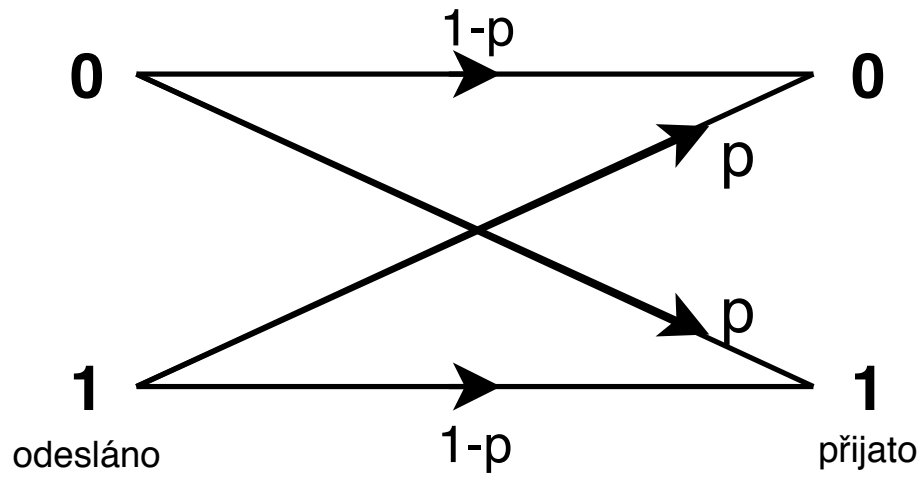
Komunikačním kanálem budeme rozumět tzv. **q-ární symetrický kanál**. Jedná se o model zašuměného komunikačního kanálu, kde každý z q používaných symbolů má stejnou pravděpodobnost p ($p < \frac{1}{2}$), že při jeho přenosu dojde k chybě. Zároveň pokud symbol dorazí s chybou,

pro každý ze zbývajících $q-1$ symbolů je stejná pravděpodobnost, že se původní symbol změnil právě na něj.

Symetrický kanál je teoretický model komunikačního kanálu a při práci s ním se obecně neuvažuje možnost, kdy se symbol úplně vytratí. Připouští se pouze možnost, že se symbol změní na jiný.

V této práci se budeme převážně zabývat **binárními kódy**, tzn. kódy, jejichž abeceda je tvořena pouze symboly 0 a 1 ($F_2 = \{0, 1\}$) [2].

Schéma binárního symetrického kanálu vypadá následovně:



Obrázek 1: Binární symetrický kanál

Pokud uvažujeme binární symetrický kanál, kterým jsme odeslali kódové slovo délky n, bude pravděpodobnost, že nedošlo k žádným chybám $(1-p)^n$. Pravděpodobnost, že jedna chyba nastane na předem určené pozici je pak $p(1-p)^{n-1}$ a pravděpodobnost, že přijaté slovo obsahuje chyby přesně na i předem určených pozicích je $p^i(1-p)^{n-i}$. Pro $p < \frac{1}{2}$ je zřejmé, že nejpravděpodobnější možností je, že nedojde k žádné chybě.

Obecně se připouští, aby jednotlivá kódová slova neměla stejnou délku (např. z obecných kódů již zmíněný *Morseův kód*, ze samoopravných kódů např. *konvoluční kódy* [2]). Skupinou kódů se kterou budeme převážně pracovat jsou tzv. blokové kódy.

Definice 1 *Blokový kód je takový kód, jehož všechna kódová slova mají stejnou délku.*

Množinu n-tic (slov délky n) tvořených symboly abecedy F_q označíme $(F_q)^n$. Pokud libovolné kódové slovo blokového kódu chápeme jako vektor $\vec{c} = (c_1, c_2, ..., c_n)$, pak $\vec{c} \in (F_q)^n$, kde $c_1, c_2, ..., c_n \in F_q$.

Pokud od kódu očekáváme jisté vlastnosti (hlavně opravování chyb), musíme zpřísňovat i požadavky, které na množinu kódových slov klademe. Proto nevystačíme s libovolným binárním blokovým kódem.

Samoopravné kódy fungují na principu tzv. **redundance**, což je vhodné prodloužení původní zprávy tak, aby jednotlivá kódová slova byla od sebe natolik různá, že i po záměně několika málo symbolů vlivem chyby s jistotou víme, že se nemohlo jednat o žádné jiné kódové slovo.

Jak jsou od sebe jednotlivá kódová slova zvoleného blokového kódu různá chápeme ve smyslu Hammingovy vzdálenosti.

Definice 2 *Hammingova vzdálenost dvou kódových slov $\vec{x}$ a $\vec{y}$, kde $\vec{x}, \vec{y} \in (F_q)^n$, je rovna počtu pozic (souřadnic), ve kterých se liší. Značíme ji $d(\vec{x}, \vec{y})$.*

Příklad 3
Mějme binární blokový kód $C = \{000, 111\}$, tzv. binární opakovací kód. Pak Hammingova vzdálenost mezi oběma kódovými slovy $d(000, 111)$ je 3. ■

Čím jsou od sebe jednotlivá slova kódu vzdálenější, tím jsou různější a tím spíše malá chyba zapříčiní vznik slova, které neodpovídá žádnému z kódových slov. Budeme proto schopni odhalit, že k chybě došlo. Pokud navíc chyba slovo pozmění pouze nepatrně, budeme si i schopni domyslet, o jaké slovo se jednalo původně. Tomuto odhalování a opravování chyb na základě podobnosti (nejmenší vzdálenosti) s kódovými slovy se říká **dekódování k nejbližšímu sousedovi** (angl. *nearest neighbour decoding*). Tento způsob dekódování má smysl používat, pokud je konzistentní s dekódováním s **maximální věrohodností** (angl. *maximum likelihood decoding*), které hledá kódové slovo $\vec{y}$ tak, aby pravděpodobnost, že bylo přijato slovo $\vec{x}$ za podmínky, že bylo odesláno kódové slovo $\vec{y}$, byla maximální. V binárním symetrickém kanálu si ovšem oba dekódovací přístupy odpovídají.

Je zřejmé, že celková různost kódových slov bude záviset na "nejslabším článku" (dvou nejméně různých kódových slovech). Hovoříme proto o tzv. minimální vzdálenosti kódu.

Definice 3 *Minimální vzdálenost d kódu C je nejmenší Hammingova vzdálenost mezi jednotlivými kódovými slovy, tzn.*

$$d(C) = \min \{d(\vec{x}, \vec{y}) \mid \vec{x}, \vec{y} \in C, \vec{x} \neq \vec{y}\}.$$

Věta 1 [3] Kód C s minimální vzdáleností d opraví až t chyb, pokud platí $d \geq 2t + 1$, kde $d, t \in \mathbb{N}$.

Důkaz předchozího tvrzení je uveden v bakalářské práci [13], zde již uváděn nebude.

Důsledkem předchozí věty je, že pokud je minimální vzdálenost d kódu C sudá, kód opraví $t = \frac{d-2}{2}$ chyb. Pokud je lichá, kód opraví $t = \frac{d-1}{2}$ chyb. Jednotně můžeme s použitím dolní celé části zapsat, že kód s minimální vzdáleností d (sudou či lichou) opraví $t = \lfloor \frac{d-1}{2} \rfloor$ chyb.

Nyní zaveďme jednotné značení, které budeme užívat po zbytek práce. Minimální vzdálenost kódu již byla zavedena výše, značíme ji d nebo $d(C)$. Délku zprávy, kterou budeme chtít odesílat označme k a celkovou délku kódového slova po přidání redundance označme n. Počet chyb, které je kód schopen opravit, označme t a celkový počet kódových slov M.

Definice 4 *Kód C s délkou kódového slova n, délkou zprávy k a minimální vzdáleností d nazveme (n,k,d)-kód.*

Další charakteristikou, podle které se kódy srovnávají, je informační poměr.

Definice 5 *Informační poměr (n, k, d)-kódu C se značí $R(C)$ a je definován jako $\frac{k}{n}$.*

Čím vyšší je informační poměr kódu, tím méně "zbytečných" symbolů kódová slova obsahují. Čím méně jich však obsahují, tím jsou si podobnější (tím menší je minimální vzdálenost), a tím méně opraví chyb. Najít rovnováhu mezi těmito vzájemně protichůdnými veličinami je jedním z klíčových problémů teorie kódování. Od kódu bychom totiž pochopitelně chtěli, aby opravil co nejvíce chyb, obsahoval co nejvíce kódových slov, ale zároveň nebyl příliš dlouhý. Snažíme se tedy optimalizovat parametry d, M a n. Pokud pevně zvolíme dva z těchto parametrů a snažíme se k nim nalézt co možná nejlepší třetí, řešíme tzv. **hlavní problém teorie kódování**. Jeho nejběžnější variantou je nalézt kód s největším počtem slov M pro pevně zvolené n a d.

2.2 Odhady počtu slov samoopravných kódů

Na úvod této podkapitoly zaveďme další standardně užívané značení. Největší možný počet slov M takový, že existuje q-ární (n, k, d)-kód s M slovy označme $A_q(n, d)$ [3], [5].

O tom, kolik kódových slov může maximálně obsahovat (n, k, d)-kód, hovoří následující odhady. Prvním z nich je **Hammingův**.

Věta 2 *(Hammingův odhad)* [2–5] Kód C s délkou kódového slova n, počtem slov M, abecedou o velikosti q a schopností opravit t chyb splňuje

$$M\left\{\binom{n}{0} + \binom{n}{1}(q-1) + \ldots + \binom{n}{t}(q-1)^t\right\} \leq q^n. \tag{1}$$

Důkaz [3] Pokud kód opraví t chyb, pak pro každé kódové slovo existuje množina nekódových slov stejné délky, které se s daným kódovým slovem liší na nejvýše t pozicích. Tyto množiny

musí být vzájemně disjunktní, jinak by kód nemohl opravit t chyb. Počet prvků v jedné takové množině musí být

$$\binom{n}{0} + \binom{n}{1}(q-1) + \ldots + \binom{n}{t}(q-1)^t,$$

protože se zde nachází slova, která se s daným kódovým slovem liší o nula pozic (těch bude $\binom{n}{0}$), o jednu pozici (těch bude $\binom{n}{1}(q-1)$), až nakonec o t pozic ($\binom{n}{t}(q-1)^t$). Každé kódové slovo musí mít svou množinu, proto celkový počet slov v množinách bude

$$M\left\{\binom{n}{0} + \binom{n}{1}(q-1) + \ldots + \binom{n}{t}(q-1)^t\right\}.$$

Toto číslo musí zcela jistě být menší nebo rovno než maximální počet slov délky n, které lze vytvořit abecedou tvořenou q symboly. Proto

$$M\left\{\binom{n}{0} + \binom{n}{1}(q-1) + \ldots + \binom{n}{t}(q-1)^t\right\} \leq q^n.$$

■

Pro binární kódy můžeme Hammingův odhad přepsat jako:

$$M\left\{1 + \binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{t}\right\} \leq 2^n, \tag{2}$$

a tudíž pro maximální počet slov binárního kódu platí:

$$A_2(n,d) \leq \frac{2^n}{\{1 + \binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{t}\}}. \tag{3}$$

Kódy, které v Hammingově odhadu dosahují rovnosti, se nazývají **perfektní kódy**. Speciální třídou perfektních kódů jsou **Hammingovy kódy**. Jedná se o třídu lineárních kódů (vysvětleno později v kapitole 2.3), které mají minimální vzdálenost 3, a tedy opraví jednu chybu. Další třídou perfektních kódů jsou již zmíněné opakovací kódy. O opakovacích kódech se někdy říká, že jsou triviálně perfektní. Další triviálně perfektní kódy jsou pak $(n, n, 1)$-kód bez redundance a $(n, 1, 2n{+}1)$-kód, který obsahuje pouze nulový vektor délky n (minimální vzdálenost takovéhoto kódu si můžeme zvolit libovolnou, protože pro kód s jedním kódovým slovem není definovaná) [5].

Zároveň je pro upřesnění dobré zdůraznit, že z věty 1 vyplývá, že kód s minimální vzdáleností $d = 2t + 1$ opraví stejný počet chyb t jako kód s minimální vzdáleností $d = 2t + 2$. Hammingův odhad proto bude vycházet stejně jak pro $(n, k, 2t + 1)$-kód, tak i pro $(n, k, 2t + 2)$-kód.

Další odhad pro maximální počet slov kódu je **Singletonův** odhad.

Věta 3 *(Singletonův odhad)* [2–5] Pro kód C s délkou kódového slova n, minimální vzdáleností d a abecedou o velikosti q platí

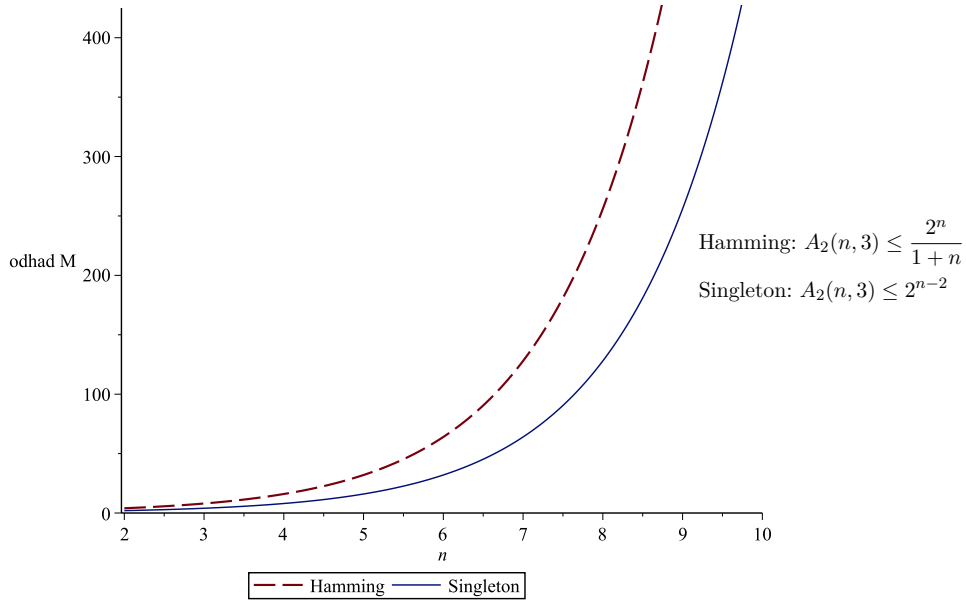
$$A_q(n,d) \leq q^{n-d+1} \tag{4}$$

Důkaz [3] Mějme abecedu o velikosti q. Maximální počet různých slov délky n, které můžeme s touto abecedou sestavit, je zjevně q^n. Dále pro libovolný (n,k,d)-kód C definovaný nad touto abecedou $(F_q)^n$ platí, že všechna kódová slova jsou určitě různá. Pokud tato slova zkrátíme o prvních $d-1$ symbolů, nově vzniklá slova musí pořád zůstat různá, protože jinak by kód neměl minimální vzdálenost d. Zkrácením slov jsme nezměnili jejich počet, ale pouze délku, která je nyní $n-(d-1) = n-d+1$. Slov tudíž nemůže být více než q^{n-d+1}. ∎

Kódy, které v Singletonově odhadu dosahují rovnosti, se nazývají **MDS kódy** (**maximum distance separable**). Jediné binární MDS kódy jsou triviální $(n,n,1)$-kódy bez redundance, opakovací $(n,1,n)$-kódy a $(n,n-1,2)$-kódy, které vzniknou přidáním jednoho tzv. paritního bitu [6], [9]. Pokud bychom chtěli složitější MDS kódy, museli bychom zvolit jinou abecedu. Například Reed-Solomonovy kódy jsou MDS kódy, které nejčastěji využívají abecedu tvořenou konečným tělesem o velikosti mocniny dvojky [2].

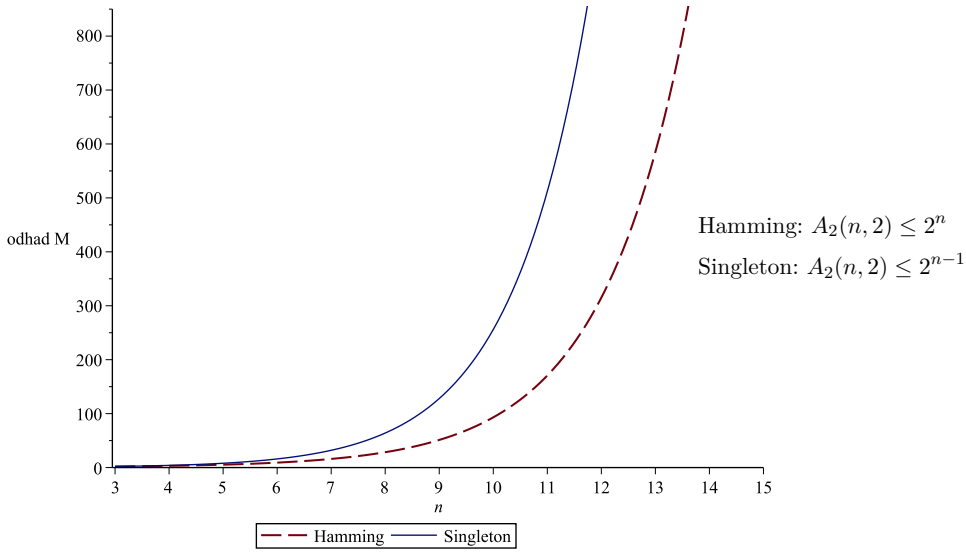
Nyní se budeme zabývat srovnáním Hammingova a Singletonova odhadu. Například by nás mohlo zajímat, zda pokud kód dosahuje rovnosti v jednom odhadu, je tomu tak i ve druhém odhadu (tzn. jestli když je kód perfektní, je zároveň i MDS a naopak). Dosaďme nyní do obou odhadů parametry konkrétních kódů. Singletonův odhad maximálního počtu slov pro binární $(3,2,2)$-kód vychází $A_2(3,2) \leq 4$. Hammingův odhad pro stejné parametry vychází $A_2(3,2) \leq 8$. Kód má 4 kódová slova, dosahuje tudíž rovnosti pouze v Singletonově odhadu. Je proto MDS, ale není perfektní. Naproti tomu pro binární $(7,4,3)$-kód vychází Hammingův odhad $A_2(7,3) \leq 16$ a Singletonův odhad $A_2(7,3) \leq 32$. Kód má 16 kódových slov, a tedy dosahuje rovnosti pouze v Hammingově odhadu. Je proto perfektní, ale není MDS. A nakonec pro binární opakovací $(3,1,3)$-kód vycházejí oba odhady $A_2(3,3) \leq 2$ a kód má skutečně dvě kódová slova. Rovnost nastává v obou odhadech, kód je proto současně perfektní i MDS. Je tedy vidět, že dosáhne-li nějaký kód maximálního počtu slov v jednom z odhadů, neznamená to, že druhý odhad nutně dává stejný výsledek.

Dále nás může zajímat, zda se dá obecně nebo alespoň pro specifikovanou množinu kódů prohlásit, že jeden odhad je přísnější než druhý. Zafixujme nyní $q = 2$ a $d = 3$ (tzn. $t = 1$) a vynesme do grafu výsledky pro odhad $A_q(n,d)$ při proměnlivém n (graf pro přehlednost nechme spojitý, ačkoliv má smysl uvažovat pouze celá n).



Obrázek 2: Odhad počtu slov pro $q = 2$ a $d = 3$

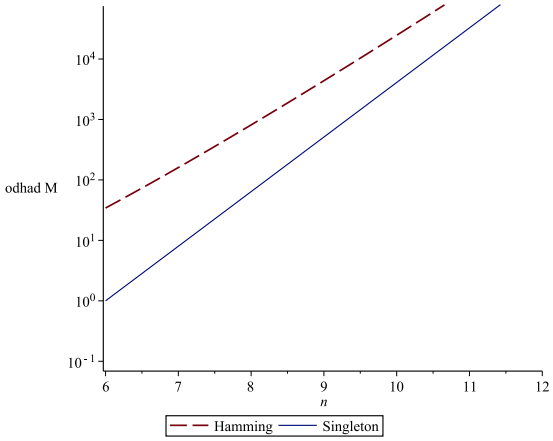
Pro $q = 2$ a $d = 3$ je zjevně přísnější Hammingův odhad. Nyní zafixujme $q = 2$ a $d = 2$ (tzn. $t = 0$).



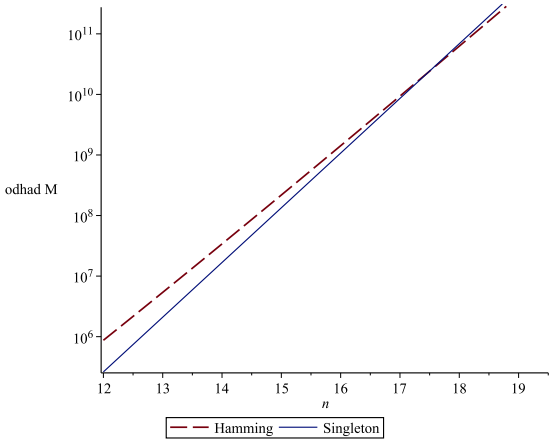
Obrázek 3: Odhad počtu slov pro $q = 2$ a $d = 2$

Pro parametry $q = 2$ a $d = 2$ je přísnější Singletonův odhad. Přísnost odhadů zjevně závisí na zvoleném parametru d.

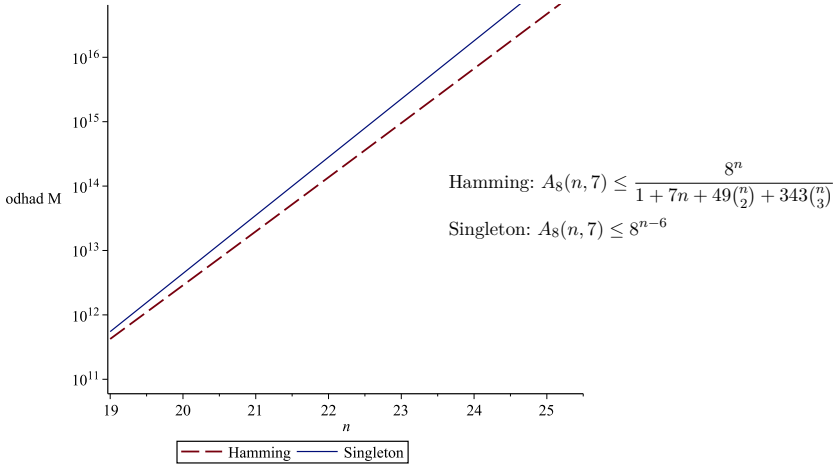
Nyní zafixujme $q = 8$, $d = 7$ (tzn. $t = 3$) a rozdělme n na tři intervaly. Jelikož očekáváme velice prudký růst, pro přehlednost grafy uveďme v logaritmickém měřítku.



Obrázek 4: Odhad počtu slov pro $q = 8$, $d = 7$ a $n \in \langle 6, 12 \rangle$



Obrázek 5: Odhad počtu slov pro $q = 8$, $d = 7$ a $n \in \langle 12, 19 \rangle$



Obrázek 6: Odhad počtu slov pro $q = 8$, $d = 7$ a $n \in \langle 19, 25 \rangle$

Na grafech můžeme pozorovat, že kolem $n = 17$ dochází k protnutí a k záměně přísnějšího odhadu, přísnost tudíž zjevně závisí i na n. Obecně tedy nemůžeme prohlásit, který z odhadů je přísnější a má proto smysl používat oba.

Existují i další odhady. Jeden z nich, tzv. **Gilbert-Varshamovův** odhad, nám například dává dolní mez maximálního počtu slov kódu s parametry n, d a q.

Věta 4 *(Gilbert-Varshamovův odhad pro obecné kódy)* [3] Mějme q-ární kód C délky n s minimální vzdáleností d. Pak pro maximální počet kódových slov $A_q(n, d)$ platí

$$A_q(n,d) \geq \frac{q^n}{\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \ldots + \binom{n}{d-1}(q-1)^{d-1}}. \tag{5}$$

Důkaz [3] Mějme kód C délky n s minimální vzdáleností d. Dále předpokládejme, že kód C má maximální možný počet slov, tzn. $M = A_q(n, d)$. Pro libovolné $\vec{x} \in (F_q)^n$ existuje alespoň jedno kódové slovo $\vec{c} \in C$ takové, že Hammingova vzdálenost $d(\vec{x}, \vec{c}) \leq d - 1$. Jinak bychom mohli $\vec{x}$ přidat do C, což je v rozporu s předpokladem, že C má maximální možný počet slov. Každému kódovému slovu proto náleží množina obsahující všechna slova, která jsou od něj ve vzdálenosti $d - 1$ nebo menší. Sjednocením množin zjevně dostaneme celé $(F_q)^n$. Velikost každé množiny je

$$\binom{n}{0} + \binom{n}{1}(q-1) + ... + \binom{n}{d-1}(q-1)^{d-1},$$

protože obsahují vždy postupně slova, která se s daným kódovým slovem liší na 0 až $d-1$ pozicích. Jelikož množiny nemusejí být vždy disjunktní (slova se v nich mohou opakovat) a celkový počet všech možných slov je q^n, tak platí, že

$$q^n \leq M \left(\binom{n}{0} + \binom{n}{1}(q-1) + ... + \binom{n}{d-1}(q-1)^{d-1} \right).$$

A protože předpokládáme, že $M = A_q(n, d)$, tak z předchozího vztahu snadno odvodíme, že

$$A_q(n, d) \geq \frac{q^n}{\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + ... + \binom{n}{d-1}(q-1)^{d-1}}.$$

■

Pro binární kódy můžeme odhad zjednodušit jako:

$$A_2(n, d) \geq \frac{2^n}{\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + ... + \binom{n}{d-1}}. \tag{6}$$

Gilbert-Varshamovův odhad má i druhou (silnější) verzi, která nám kromě dolního odhadu $A_q(n, d)$ může také pomoci potvrdit, zda kód s danými parametry existuje. Tato verze ovšem platí pouze pro lineární kódy a vrátíme se k ní proto později.

2.3 Lineární kód

Kód použitý v našem experimentu je lineární a použité kódovací a dekódovací algoritmy vycházejí z vlastností lineárních kódů, proto je potřeba shrnout tuto základní skupinu samoopravných kódů. Hlavní definice a vlastnosti lineárních kódů jsou součástí téměř každého učebního textu o samoopravných kódech [2–5].

V další části textu značením $V(n, q)$ budeme rozumět vektorový prostor definovaný nad konečným tělesem $GF(q)$, kde $n \in \mathbb{N}$ určuje dimenzi prostoru a q je prvočíslo či mocnina prvočísla představující počet symbolů abecedy.

Definice 6 *Kód C, jehož kódová slova tvoří vektorový podprostor ve vektorovém prostoru $V(n, q)$, se nazývá lineární.*

Lineární kód je tedy vždy uzavřený vůči sčítání vektorů (kódových slov) a vůči násobení skalárem a vždy obsahuje nulový vektor jako jedno z kódových slov. Jelikož pracujeme hlavně s binárními kódy, násobení skalárem není třeba ověřovat (bude vždy splněno). Pokud tedy chceme ověřit, zda je daný binární kód lineární, stačí se podívat, zda obsahuje nulový prvek a zda platí, že součtem libovolných dvou kódových slov dostaneme jiné kódové slovo. Abecedu F_2 jsme zkonkretizovali na konečné těleso $GF(2)$, sčítání a násobení je tudíž myšleno po složkách modulo 2.

Jako lineární kód můžeme chápat i celý prostor $V(n, q)$. Takový kód je $(n, n, 1)$-kód, a tudíž, jak už bylo zmíněno, triviálně perfektní (tak jako kód tvořený pouze nulovým vektorem).

Stejně jako u obecného kódu i zde můžeme konkrétní kódy označovat jako (n, k, d)-kódy. Lineární kódy v tomto textu budeme od obecných kódů rozlišovat užitím hranatých závorek, tj. budeme hovořit o $[n, k, d]$-kódech. Případně můžeme informaci o minimální vzdálenosti vynechat a hovořit pouze o $[n, k]$-kódu.

Příklad 4
Kód $C = \{000, 011, 101, 110\}$ je lineární $[3, 2, 2]$-kód. ■

Jednu z výhod lineárních kódů udává následující věta. Nejprve ovšem potřebujeme zavést další pojem, kterým je váha vektoru.

Definice 7 *Mějme libovolný vektor $\vec{a}$. Hammingova váha $w(\vec{a})$ je počet nenulových souřadnic vektoru $\vec{a}$.*

Příklad 5
Pro $\vec{a} = (1, 0, 1)$ platí, že $w(\vec{a}) = 2$. ■

Zmíněná věta je pak následující.

Věta 5 *Minimální vzdálenost d lineárního kódu je rovna nejmenší váze nenulových kódových slov kódu C.*

I tento důkaz je uveden v bakalářské práci [13].

Dalším důležitým pojmem, týkajícím se lineárních kódů, je generující matice. Jednou z výhod lineárních kódů totiž je, že pro plnou specifikaci nemusíme vypisovat všechna kódová slova, stačí nám pouze uvést generující matici.

Definice 8 *Matice G o velikosti $k \times n$, jejíž řádky tvoří bázi lineárního $[n, k]$-kódu, se nazývá generující matice.*

Příklad 6
Generující matice lineárního $[3, 2, 2]$-kódu z předchozího příkladu 4 je

$$G = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}.$$

■

Generující matice jednoznačně určuje lineární kód, ke kterému náleží. Každé kódové slovo příslušného kódu lze vyjádřit jako lineární kombinaci jejích řádků. Generující matici potřebujeme hlavně pro kódování zpráv. Zakódovat zprávu $\vec{m}$ do kódového slova $\vec{c}$ lineárního kódu znamená vynásobit jí zprava generující maticí G.

Příklad 7
Opět uvažujme $[3, 2, 2]$-kód $C = \{\vec{c} \in V(3, 2) : \vec{c} = \vec{m} \cdot G \wedge \vec{m} \in V(2, 2)\}$, který kóduje zprávy 00,01,10 a 11. Zprávě $\vec{m} = (0, 1)$ odpovídá kódové slovo $\vec{c} = (1, 0, 1)$, které obdržíme výpočtem:

$$\vec{m} \cdot G = \begin{bmatrix} 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} = \vec{c}.$$

■

2.3.1 Dekódování pomocí standard array

Než přejdeme k dekódování, musíme nejprve uvést potřebné definice a věty.

Definice 9 *Mějme $[n, k]$-kód C nad abecedou $GF(q)$ a libovolný vektor $\vec{a} \in V(n, q)$. Množinu $\vec{a} + C = \{\vec{a} + \vec{x} \mid \vec{x} \in C\}$ nazveme coset kódu C.*

Slovo coset je převzato z angličtiny. Český ekvivalent není, nejblíže odpovídá pojem "třída rozkladu grupy", proto zůstaneme u anglického názvu. Čteme "koset" a skloňujeme dle vzoru hrad.

Následující vlastnosti cosetů jsou důležité pro dekódování a jejich důkazy lze najít ve většině učebnic základních kurzů algebry nebo kurzů o samoopravných kódech [2–5]. Případně jsou oba důkazy uvedeny i v bakalářské práci [13].

Věta 6 *Mějme coset $\vec{a} + C$ a prvek $\vec{b} \in \vec{a} + C$. Pak $\vec{a} + C = \vec{b} + C$.*

Věta 7 *(Lagrangeova) Mějme $[n, k]$-kód C nad abecedou $GF(q)$. Pak*

1. *každý vektor z $V(n, q)$ patří do nějakého cosetu kódu C,*
2. *každý coset obsahuje přesně q^k vektorů,*
3. *dva cosety se nikdy částečně nepřekrývají (mají společné buď všechny prvky nebo žádný prvek), tj. tvoří rozklad na množině všech kódových slov.*

Příklad 8
Mějme opět $[3, 2, 2]$-kód. Rozklad vektorového prostoru $V(3, 2)$ na cosety je:

$$000 + C = 011 + C = 101 + C = 110 + C = \{000, 011, 101, 110\} = C,$$
$$100 + C = 111 + C = 001 + C = 010 + C = \{100, 111, 001, 010\}.$$

■

Prvku s nejmenší váhou v cosetu říkáme **reprezentant cosetu**. Pokud je prvků s nejmenší váhou v cosetu více, můžeme za reprezentanta zvolit libovolný z nich.

Příklad 9
Reprezentantem cosetu $\{100, 111, 001, 010\}$ můžeme nazvat prvek 100, 001 nebo 010. ■

Jedním ze způsobů dekódování lineárních kódu, který využívá cosetů, je dekódování pomocí tzv. **standard array**. Standard array $[n, k]$-kódu C je tabulka o rozměrech $q^{n-k} \times q^k$, jejíž prvky jsou všechny vektory prostoru $V(n, q)$. První řádek tvoří samotný kód C, nulový vektor je vždy uveden jako první. Zbylé řádky jsou cosety, kdy v prvním sloupci je uveden reprezentant.

Postup pro vytvoření standard array je následující:

1. Na první řádek vypíšeme všechna kódová slova kódu C. Jako první uvedeme nulový vektor, zbytek můžeme zapsat v libovolném pořadí.

2. Zvolíme libovolný vektor $\vec{a_1} \in V(n, q)$, který má minimální váhu a nenachází se v prvním řádku. Zapíšeme jej jako první. Zbytek řádku bude tvořen cosetem $\vec{a_1} + C$. Pořadí je určeno tak, že vektor $\vec{a_1} + \vec{x}$, kde $\vec{x} \in C$, je v tabulce zapsán pod vektor $\vec{x}$.

3. Zvolíme libovolný vektor $\vec{a_2} \in V(n, q)$, který má minimální váhu a nenachází se v prvním ani druhém řádku. Zbytek řádku bude tvořen cosetem $\vec{a_2} + C$. Pořadí je určeno stejně jako v předešlém kroku.

4. Pokračujeme stejným postupem tak dlouho, dokud nemáme vypsány všechny cosety a každý vektor z $V(n, q)$ není uveden právě jednou.

Příklad 10
Standard array $[3, 2, 2]$-kódu C je například:

000	**011**	**101**	**110**
100	111	001	010

Tabulka 1: Standard array $[3, 2, 2]$-kódu

■

Dekódování pomocí standard array probíhá tak, že přijaté slovo $\vec{r}$ jednoduše nalezneme v tabulce a podíváme se, jaké kódové slovo $\vec{c}$ se nachází ve stejném sloupci na prvním řádku. Na toto slovo dekódujeme. Slova úplně vlevo v tabulce (reprezentanti) představují chybové vektory $\vec{e}$. Vektor $\vec{e}$ představuje vzor chyby, tzn. k jaké chybě v přijatém slově nejspíše došlo. V podstatě tedy vyhledáváním v tabulce provádíme výpočet $\vec{c} = \vec{r} - \vec{e}$.

Použitý [3,2,2]-kód má minimální vzdálenost $d = 2$, nelze proto použít k opravě libovolné jedné chyby (pouze k detekci). Pomocí námi sestaveného standard array ovšem můžeme opravit chybový vzor $\vec{e} = (1, 0, 0)$. Pokud tedy chyba nastane na první pozici, opravíme ji správně. Pokud nastane na druhé nebo třetí pozici, chybový vzor bude jiný, a tudíž dekódujeme špatně. Zároveň je dobré zmínit, že podoba námi sestaveného standard array není jediná možná. Jako opravitelný chybový vzor jsme mohli zvolit také 010 nebo 001. Jelikož ovšem všechny tři chybové vzory náleží do stejného cosetu, můžeme zvolit vždy jenom jeden z nich. U $[3, 2, 2]$- kódu tedy chybu opravíme vždy jen na právě jedné předem dané pozici.

Nyní se budeme zabývat tím, jak spolehlivé je dekódování pomocí standard array pro binární $[n, k]$-kódy.

Věta 8 [3] Mějme binární $[n, k]$-kód C. Dále pro každé $i = 0, ..., n$ mějme číslo α_i představující počet reprezentantů váhy i. Pak pravděpodobnost P_{corr}, že přijaté slovo dekódované pomocí standard array skutečně odpovídá odeslanému kódovému slovu je

$$P_{corr}(C) = \sum_{i=0}^{n} \alpha_i p^i (1-p)^{n-i}. \tag{7}$$

Důkaz Platnost uvedeného vztahu přímo plyne z faktu, že pravděpodobnost, že přijaté slovo obsahuje chyby přesně na i předem daných pozicích je v binárním symetrickém kanálu s pravděpodobností chyby p

$$p^i (1-p)^{n-i}.$$

Tato pravděpodobnost je ekvivalentní s pravděpodobností, že chybový vektor je vektor váhy i. ■

Příklad 11
Lineární $[3, 2, 2]$-kód C má dva reprezentanty 000 a 100, tudíž $\alpha_0 = 1$, $\alpha_1 = 1$ a $\alpha_2 = \alpha_3 = 0$. Proto

$$P_{corr}(C) = (1-p)^3 + p(1-p)^2 = (1-p)^2((1-p)+p) = (1-p)^2.$$

Pro $p = 0,01$ je $P_{corr}(C) = 0,9801$. ■

Pravděpodobnost, že dekódované slovo není slovo, které bylo odesláno (chybné dekódování), je

$$P_{err}(C) = 1 - P_{corr}. \tag{8}$$

Příklad 12
Když použijeme výsledek předchozího příkladu, dostaneme pro $p = 0,01$

$$P_{err}(C) = 1 - 0,9801 = 0,0199.$$

■

Pravděpodobnost, že nezakódovaná zpráva délky 2 dorazí binárním symetrickým kanálem s chybou, je pro $p = 0,01$

$$1 - (1-p)^2 = 0,0199,$$

tudíž chybně dorazí úplně se stejnou pravděpodobností, jako kdyby byla zakódována lineárním $[3,2,2]$-kódem. Pokud nám jde o opravování chyb a ne jen o detekci, použití $[3,2,2]$-kódu se zjevně nevyplatí, protože jsme zprávu prodloužili a pravděpodobnost zůstala nezměněna. Výsledek není nijak překvapující, protože kód není určen k opravování chyb.

Zajímavé však je, že pokud přidáme ještě jeden redundantní bit a sestavíme lineární $[4,2,2]$-kód $C = \{0000, 1011, 0101, 1110\}$, minimální vzdálenost d zůstane nezměněna a kód stále nebude schopen opravit chybu na libovolné pozici. Přesto pokud sestavíme standard array, získáme schopnost opravit chyby již na třech pozicích ze čtyř.

0000	**1011**	**0101**	**1110**
1000	0011	1101	0110
0100	1111	0001	1010
0010	1001	0111	1100

Tabulka 2: Standard array $[4,2,2]$-kódu

Nyní máme 4 reprezentanty. Konstanty α_i jsou $\alpha_0 = 1$, $\alpha_1 = 3$ a $\alpha_2 = \alpha_3 = \alpha_4 = 0$. Pravděpodobnost P_{corr} je

$$P_{corr}(C) = (1-p)^4 + 3p(1-p)^3 = (1-p)^3(1+2p),$$

a tudíž pro $p = 0,01$ bude

$$P_{err}(C) = 1 - 0,9897 = 0,0103.$$

Přidáním dalšího bitu navíc jsme pravděpodobnost, že dojde k chybnému dekódování snížili přibližně dvojnásobně.

2.3.2 Syndromové dekódování

Pro malé kódy je dekódování pomocí standard array ideální volbou, tabulka je jednoduchá na sestavení a snadný je i samotný proces vyhledávání. V praxi ovšem pracujeme s mnohonásobně

většími kódy, které čítají tisíce až miliony kódových slov. Je zřejmé, že uchovávat v paměti tak obrovskou tabulku (pokud je to vůbec možné) je značně neefektivní. Neefektivní je i proces vyhledávání, protože v nejhorším případě musíme prohledat celou tabulku. Časová složitost vyhledávání tudíž roste kvadraticky s počtem prvků v tabulce (otázkou časové a paměťové složitosti se budeme zabývat ještě na konci této podkapitoly). Pokud potřebujeme pracovat s většími kódy, máme v podstatě dvě možnosti. Přejít k složitějším třídám kódů, jejichž kódovací a dekódovací algoritmy jsou za cenu složitějšího matematického aparátu časově i paměťově efektivnější, nebo (pokud trváme na použití obecného lineární kódu) můžeme zvolit jiný dekódovací algoritmus, tzv. syndromové dekódování. Než jej však budeme moci popsat, musíme nejprve definovat nové pojmy.

Definice 10 *Mějme lineární $[n, k]$-kód C. Duálním kódem, značeným $C^{\perp}$, nazveme množinu těch vektorů z $V(n, q)$, které jsou ortogonální ke každému kódovému slovu z C, tzn.*

$$C^{\perp} = \{\vec{v} \in V(n, q) \mid \vec{v} \cdot \vec{u} = 0, \ \forall \vec{u} \in C\}.$$

Výrazem $\vec{v} \cdot \vec{u}$ v předchozí definici rozumíme standardní skalární součin, tzn. pro $\vec{v} = (v_1, ..., v_n)$ a $\vec{u} = (u_1, ..., u_n)$ je

$$\vec{v} \cdot \vec{u} = \sum_{i=1}^{n} v_i u_i.$$

Věta 9 *Mějme lineární $[n, k]$-kód C nad $GF(q)$. Pak $C^{\perp}$ je také lineární kód.*

Důkaz Předpokládejme, že $\vec{v_1}, \vec{v_2} \in C^{\perp}$ a $\lambda, \mu \in GF(q)$. Pak pro každé $\vec{u} \in C$ platí

$$(\lambda\vec{v_1} + \mu\vec{v_2}) \cdot \vec{u} = \lambda(\vec{v_1} \cdot \vec{u}) + \mu(\vec{v_2} \cdot \vec{u}) = \lambda 0 + \mu 0 = 0,$$

tudíž $\lambda\vec{v_1} + \mu\vec{v_2} \in C^{\perp}$, a proto $C^{\perp}$ je lineární. ∎

Věta 10 *Duální kód $C^{\perp}$ lineárního $[n, k]$-kódu C nad $GF(q)$ má dimenzi $n - k$.*

Důkaz této věty je trochu delší a vyžaduje další pomocné tvrzení. Z tohoto důvodu pouze nastíníme základní myšlenku a odkážeme čtenáře na zdroj [3], kde je důkaz uveden celý. Princip důkazu spočívá v tom, že $C^{\perp}$ je řešením soustavy lineárních rovnic s koeficienty z generující matice G o rozměrech $k \times n$ a nulovou pravou stranou. Je tedy nulovým prostorem matice G. Standardním výsledkem lineární algebry je, že tento prostor je dimenze $n - k$, kde n je počet sloupců matice G a k je hodnost matice G.

Příklad 13
Mějme opět lineární $[3, 2, 2]$-kód $C = \{000, 011, 101, 110\}$. Jeho duální kód je $C^{\perp} = \{000, 111\}$.
∎

Jelikož je duální kód lineárního kódu taktéž lineární, musí mít svoji generující matici.

Definice 11 *Mějme $[n, k]$-kód C a jeho duální kód $C^{\perp}$. Generující matici duálního kódu $C^{\perp}$ nazveme kontrolní maticí původního kódu C. Značíme ji H.*

Kontrolní matici H můžeme definovat i alternativně bez použití duálního kódu jako matici o rozměrech $(n-k) \times n$, která splňuje rovnost $GH^T = O$, kde O je nulová matice o rozměrech $k \times (n-k)$.

Vyvstává otázka, jakým způsobem můžeme kontrolní matici nalézt. Využít můžeme generující matici G, kterou je ovšem nutno převést do standardního tvaru $G = [I_k|A]$, kde I_k je jednotková matice o rozměrech $k \times k$ a A matice o rozměrech $k \times (n-k)$.

Do požadovaného tvaru generující matici dostaneme užitím ekvivalentních řádkových úprav nebo záměnou pořadí sloupců. Pokud používáme pouze ekvivalentní řádkové úpravy, množina kódových slov zůstává stejná. Pokud používáme i sloupcové úpravy, dostáváme kód o stejných parametrech, který je ekvivalentní s původním kódem. V této práci pro nás ekvivalence kódů není až tolik důležitá, případného zájemce o tuto problematiku můžeme odkázat na zdroj [3].

Věta 11 *Mějme lineární $[n, k]$-kód C s generující maticí G ve standardním tvaru $G = [I_k|A]$. Pak kontrolní matice H má tvar $H = [-A^T|I_{n-k}]$.*

Důkaz předchozího tvrzení je uveden v bakalářské práci [13].

Pokud pracujeme s binárním kódem, znaménko mínus v předpisu pro kontrolní matici není potřeba uvažovat.

Příklad 14
Mějme opět lineární $[3, 2, 2]$-kód. Jeho generující matice

$$G = \left[\begin{array}{cc|c} 1 & 0 & 1 \\ 0 & 1 & 1 \end{array}\right]$$

se již nachází ve standardním tvaru, a proto kontrolní matice má tvar

$$H = \left[\begin{array}{cc|c} 1 & 1 & 1 \end{array}\right].$$

Trochu větší $[4, 2, 2]$-kód má generující matici

$$G = \left[\begin{array}{cc|cc} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{array}\right],$$

která je opět již ve standardním tvaru a odpovídající kontrolní matice má tvar

$$H = \left[\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{array}\right].$$

■

Důležitým prostředkem využívaným v dekódovacím algoritmu, který se chystáme popsat, je syndrom.

Definice 12 *Mějme $[n, k]$-kód C s kontrolní maticí H a libovolný vektor $\vec{r} \in V(n, q)$. Řádkový vektor $S(\vec{r}) = \vec{r}H^T$ o rozměrech $1 \times (n - k)$ nazveme syndrom vektoru $\vec{r}$.*

Klíčová vlastnost syndromu, vyplývající z vlastností kontrolní matice je, že syndromy kódových slov budou vždy nulové vektory a syndromy nekódových slov (chybných slov) budou vždy nenulové vektory. Další důležitá vlastnost syndromů souvisí s cosety.

Věta 12 *Dva vektory $\vec{u}$ a $\vec{v}$ se nacházejí ve stejném cosetu právě tehdy, když je jejich syndrom stejný.*

Důkaz předchozího tvrzení je opět uveden v bakalářské práci [13].

Tyto klíčové vlastnosti vedou k sestavení nové dekódovací tabulky, která je oproti původnímu standard array menší. Z původní tabulky totiž můžeme nechat pouze první sloupec s reprezentanty, ostatní sloupce můžeme odstranit a nahradit je jedním sloupcem syndromů reprezentantů. Bez ohledu na počet kódových slov budeme mít vždy tabulku o dvou sloupcích, která se nazývá **syndromová tabulka**.

Příklad 15
Syndromy reprezentantů $[3, 2, 2]$-kódu jsou

$$S(000) = \vec{r}H^T = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 0,$$

$$S(100) = \vec{r}H^T = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 1.$$

Syndromová tabulka vypadá následovně:

Reprezentanti	Syndromy
000	0
100	1

Tabulka 3: Syndromová tabulka $[3, 2, 2]$-kódu

Syndromy reprezentantů $[4, 2, 2]$-kódu jsou

$$S(0000) = \vec{r}H^T = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = 00,$$

$$S(1000) = \vec{r}H^T = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = 11,$$

$$S(0100) = \vec{r}H^T = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = 01,$$

$$S(0010) = \vec{r}H^T = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = 10.$$

Syndromová tabulka vypadá následovně:

Reprezentanti	Syndromy
0000	00
1000	11
0100	01
0010	10

Tabulka 4: Syndromová tabulka $[4, 2, 2]$-kódu

■

Dekódování pomocí syndromové tabulky má následující algoritmus:

1. Pro přijaté slovo $\vec{r}$ vypočteme syndrom $S(\vec{r}) = \vec{r}H^T$.
2. Výsledný syndrom nalezneme v tabulce, příslušný reprezentant na stejném řádku bude chybový vektor $\vec{e}$.
3. Dekódujeme výpočtem $\vec{c} = \vec{r} - \vec{e}$.

Příklad 16
Uvažujme $[4, 2, 2]$-kód a jeho syndromovou tabulku sestavenou v předchozím příkladě 15. Přijmeme například slovo $\vec{r} = 0111$. Jeho syndrom je

$$S(0111) = \vec{r} \cdot H^T = \begin{bmatrix} 0 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{9}$$

Syndrom odpovídá poslednímu řádku tabulky a reprezentantovi 0010. Dekódujeme tedy na $\vec{c} = 0111 - 0010 = 0101$. ■

Syndromová tabulka redukuje paměťovou náročnost dekódování. Zrychlí se i vyhledávání, protože namísto vyhledávání ve dvojrozměrné tabulce prohledáváme pouze jeden jediný sloupec. Nevýhodou je, že pro sestavení syndromové tabulky stejně musíme sestavit i standard array, abychom věděli, jaké mají jednotlivé cosety reprezentanty. Zároveň pro opravdu velké kódy je i syndromová tabulka pořád poměrně rozsáhlá.

V praxi se například využívá Reed-Solomonův $[255, 223]$-kód, definovaný nad $GF(256)$. Reed-Solomonovy kódy spadají do třídy lineárních kódů a můžeme pro ně proto teoreticky sestavit standard array a syndromovou tabulku. Podle předchozích uvedených informací ovšem víme, že rozměry standard array jsou $q^{n-k} \times q^k$ a syndromové tabulky $q^{n-k} \times 2$. Pro uvedený kód by tudíž standard array mělo rozměry $256^{32} \times 255^{223}$ a syndromová tabulka $256^{32} \times 2$, což jsou naprosto enormní tabulky a je nereálné vůbec uvažovat, že bychom je byli schopni udržovat v paměti.

Pro konkrétní výpočty paměťové náročnosti obou dekódovacích tabulek můžeme případného zájemce odkázat na bakalářskou práci [13], která demonstruje výpočet pro binární $[32, 6]$-kód.

2.3.3 Gilbert-Varshamovův odhad

V předchozí kapitole jsme se zmínili o existenci verze Gilbert-Varshamova odhadu určené pouze pro lineární kódy. Než ji však uvedeme, potřebujeme nejdříve uvést pomocné tvrzení, které využijeme v důkazu.

Věta 13 [3–5] Mějme lineární $[n, k]$-kód C nad $GF(q)$ s kontrolní maticí H. Kód C má minimální vzdálenost d právě tehdy, když libovolných $d - 1$ sloupců kontrolní matice H je lineárně nezávislých a některých d sloupců je lineárně závislých.

Důkaz [3] Z věty 5 víme, že minimální vzdálenost d kódu C je rovna nejmenší váze nenulových kódových slov. Mějme vektor $\vec{x} = (x_1, ..., x_n)$ z $V(n, q)$. Pak

$$\begin{aligned} \vec{x} \in C &\Leftrightarrow \vec{x}H^T = \vec{0} \\ &\Leftrightarrow x_1\vec{H_1} + x_2\vec{H_2} + ... + x_n\vec{H_n} = \vec{0}, \end{aligned}$$

kde $\vec{H_1}, ..., \vec{H_n}$ jsou sloupce H. Tudíž každému kódovému slovu $\vec{x}$ váhy d koresponduje d lineárně závislých sloupců H. Naproti tomu pokud by existovala množina $d-1$ lineárně závislých sloupců H, označme $\vec{H_{i_1}}, \vec{H_{i_2}}, ..., \vec{H_{i_{d-1}}}$, existovaly by skaláry $x_{i_1}, x_{i_2}, ..., x_{i_{d-1}}$ takové, že ne všechny jsou nulové a

$$x_{i_1}\vec{H_{i_1}} + x_{i_2}\vec{H_{i_2}} + ... + x_{i_{d-1}}\vec{H_{i_{d-1}}} = \vec{0}.$$

To by ovšem znamenalo, že vektor $\vec{x} = (0, ..., 0, x_{i_1}, 0, ..., 0, x_{i_2}, 0, ..., 0, x_{i_{d-1}}, 0, ..., 0)$, který má x_{i_j} na pozici i_j pro $j = 1, 2, ..., d-1$ a 0 všude jinde, by splňoval $\vec{x}H^T = 0$ a byl by tím pádem nenulovým kódovým slovem s váhou menší než d. ∎

Nyní můžeme přejít k formulaci samotného Gilbert-Varshamova odhadu.

Věta 14 *(Gilbert-Varshamovův odhad pro lineární kódy)* [2–4] Předpokládejme, že q je mocnina prvočísla. Pak lineární q-ární $[n, k]$-kód s minimální vzdáleností alespoň d existuje, pokud je splněna nerovnost

$$\binom{n-1}{0} + \binom{n-1}{1}(q-1) + \binom{n-1}{2}(q-1)^2 + ... + \binom{n-1}{d-2}(q-1)^{d-2} < q^{n-k}. \tag{10}$$

Důkaz [3] Předpokládejme, že q, n, k a d splňují vztah 10. Dle věty 13, pokud zkonstruujeme $(n-k) \times n$ matici H nad $GF(q)$ takovou, že žádných $d-1$ sloupců není lineárně závislých, pak kód má minimální vzdálenost alespoň d a právě dokazované tvrzení platí. Položme $r = n - k$. Jako první sloupec H zvolme libovolnou nenulovou r-tici z $V(r, q)$. Jako druhý sloupec zvolme libovolnou r-tici, která není násobkem první. Podobně volíme další sloupce tak, aby vždy nový sloupec nebyl lineární kombinací žádných $d-2$ nebo méně předchozích sloupců. Máme $q-1$ možných nenulových koeficientů. Když chceme vybrat i-tý sloupec, r-tice, které již nemůžeme přidat, budou lineární kombinací $d-2$ nebo méně sloupců z již vybraných $i-1$ sloupců. Jejich počet bude

$$N(i) = 1 + \binom{i-1}{1}(q-1) + \binom{i-1}{2}(q-1)^2 + ... + \binom{i-1}{d-2}(q-1)^{d-2}. \tag{11}$$

Ne všechny tyto lineární kombinace musejí být různé vektory, ale i v nejhorším případě, kdy opravdu jsou, pokud $N(i)$ je menší než celkový počet všech možných r-tic (tzn. q^r), pak i-tý sloupec půjde přidat do matice. Proto, pokud vztah 10 platí, jsme schopni přidat až n-tý sloupec, a tudíž kód s danými parametry existuje. ∎

Pro binární kódy můžeme odhad přepsat jako

$$\binom{n-1}{0} + \binom{n-1}{1} + \binom{n-1}{2} + ... + \binom{n-1}{d-2} < 2^{n-k}. \tag{12}$$

Z uvedeného odhadu lze odvodit dolní mez pro maximální počet kódových slov pro lineární kód s parametry n, d a q.

Po jednoduchých algebraických úpravách jsme schopni převést vztah 10 na vztah

$$q^k < \frac{q^n}{\binom{n-1}{0} + \binom{n-1}{1}(q-1) + \binom{n-1}{2}(q-1)^2 + ... + \binom{n-1}{d-2}(q-1)^{d-2}}. \tag{13}$$

Pokud vybereme největší možné k takové, že vztah 13 platí a označíme jej $k^{'}$, pak pro největší možný počet slov $A_q(n, d)$ bude platit, že

$$A_q(n, d) \geq q^{k^{'}}. \tag{14}$$

2.3.4 Sestavení lineárního kódu

Nyní se budeme zabývat tím, jak lineární kódy konstruovat. Zkonstruovat libovolný lineární kód není složité, stačí si vzít jakýkoliv vektorový podprostor vektorového prostoru $V(n, q)$ nad $GF(q)$. Jak ale zajistit, aby daný podprostor měl přesně takovou minimální vzdálenost, jakou požadujeme? Pokud se omezíme na $d = 3$ a stačí nám tudíž kódy schopné opravovat jednu libovolnou chybu, řešení nabízí již zmíněná třída Hammingových kódů. Tyto kódy se konstruují nikoliv přes generující matici, ale přes kontrolní matici a samotný algoritmus konstrukce je poměrně jednoduchý. Výhodou také je, jak už bylo zmíněno dříve, že každý Hammingův kód je perfektní. Pro naše účely Hammingovy kódy konstruovat nepotřebujeme, proto se jejich konstrukcí nebudeme zabývat a odkážeme například na zdroj [3].

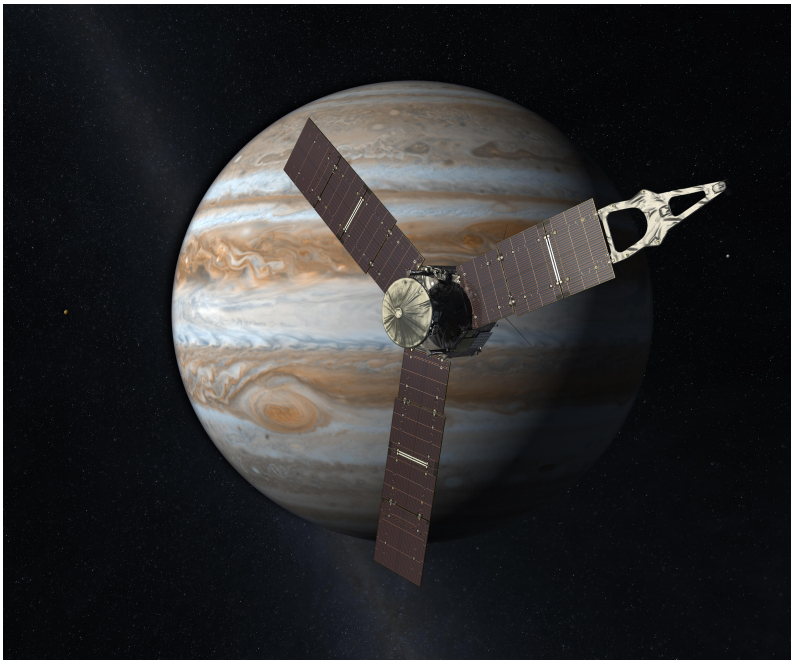
Pokud nám nestačí minimální vzdálenost $d = 3$ a požadujeme ji větší, celá situace se značně komplikuje. Zkonstruovat obecný lineární $[n, k, d]$-kód s $d > 3$ je velice obtížný úkol a často nezbývá než jej řešit "hrubou silou". V praxi se proto mnohem častěji využívají speciální podtřídy lineárních kódů (BCH-kódy, RS-kódy), kde díky mnohonásobně složitějšímu matematickému aparátu můžeme zkonstruovat a plně definovat kód s předem určenými parametry a miliony kódových slov sestavením jediného polynomu.

3 Reálná aplikace - sonda Juno

V této části si rozebereme reálnou aplikaci, využívající samoopravné kódy, která sloužila jako vzor pro náš experiment.

Obecně se se samoopravnými kódy setkáme všude, kde se přenášejí data z jednoho místa na druhé. Přenosem dat nemusíme nutně rozumět pouze internetové připojení, telefonní spojení nebo komunikaci se sondami ve Vesmíru, ale například také čtení datových nosičů (CD, DVD, Blu-Ray) nebo skenování a následnou interpretaci čárových kódů (QR-kódů). Zde všude dochází k chybám, zajistit vždy plně bezchybný přenos informací je totiž prakticky nemožný úkol. DVD je poškrábané, vizitka s QR-kódem je pokrčená, signál z Vesmíru musí proletět skrze atmosféru a spousta dalších komplikací, které mohou způsobit ztrátu informace.

Aplikací, kterou jsme nakonec zvolili, je komunikace s americkou kosmickou sondou **Juno**, jejímž hlavním cílem je zkoumání planety Jupiter (složení atmosféry, sledování jádra, magnetického pole atd.) [11]. Sonda byla zvolena hlavně kvůli aktuálnosti. Mise odstartovala v roce 2011 a sonda dorazila k oběžné dráze Jupiteru v roce 2016, kde se stále nachází. Druhým důvodem je dostupnost informací o konkrétních použitých samoopravných kódech [7]. Komunikace se sondou probíhá v obou směrech, tzn. Země posílá pokyny sondě a sonda posílá nasbíraná data zpátky na Zemi. V našem experimentu, kterému je věnována následující kapitola, je komunikace realizována pouze v prvním zmíněném směru.



Obrázek 7: Sonda Juno [19]

Sonda pro komunikaci využívá dva druhy samoopravných kódů. Prvním z nich je kombinace Reed-Solomonova kódu s konvolučním kódem. Reed-Solomonův kód, jak už bylo zmíněno, spadá do třídy lineárních kódů. Dohledaná dokumentace [7] neuvádí, jaký konkrétní Reed-Solomonův kód byl použit u sondy Juno, ale standardní RS-kód, který se používá pro komunikaci skrze Vesmír je $[255, 223]$-kód definovaný nad $GF(2^8)$ [10]. Výhodou tohoto kódu je, že se dá rozdělit na bloky o velikosti 8 bitů (1 bajt), tzn. 1 bajt představuje 1 symbol kódu. Délka kódového slova $n = 255$ zde tím pádem znamená nikoliv 255 bitů, ale $255 \cdot 8 = 2040$ bitů. Reprezentace celého bajtu jediným symbolem nám umožňuje řešit tzv. **dávkové chyby** (angl. *burst errors*), což jsou chyby, které nastávají ihned po sobě. Chyba v osmi bitech po sobě znamená maximálně dva chybné symboly. Uvedený kód je schopen opravit až 16 chybných symbolů/bajtů.

Dalším způsobem, kterým můžeme zabránit dávkovým chybám, je **prokládání** (angl. *interleaving*). Princip prokládání spočívá v rozptýlení dávkových chyb mezi jednotlivé bloky, čímž snížíme počet chyb v jednom bloku [2]. Prokládání se nejlépe demonstruje na konkrétním příkladě, který byl již uveden v bakalářské práci [13].

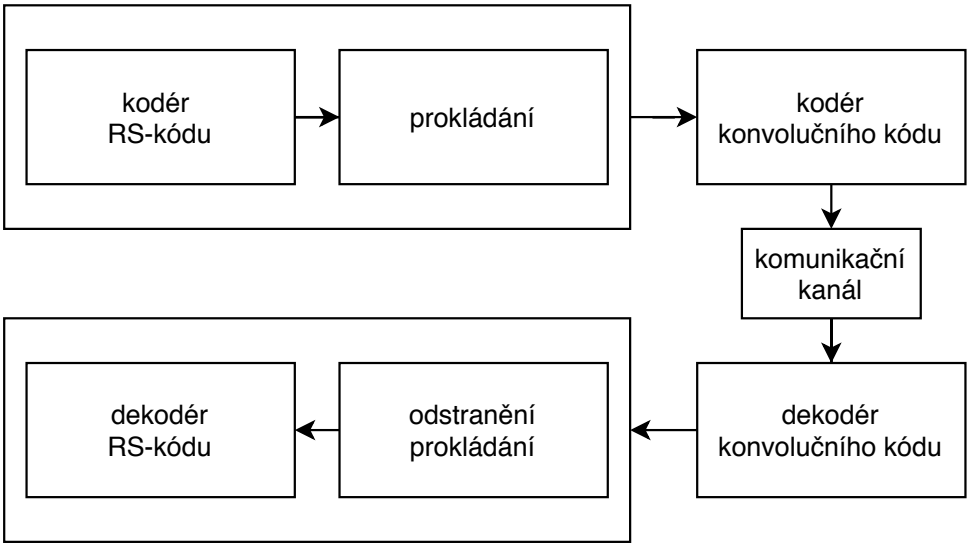
Příklad 17
Mějme konkrétní větu **TotoJePříkladProkládání**. Pokud nastanou čtyři dávkové chyby, získáme například větu **TotoJePříklad_____ládání**. Původní slovo **Prokládání** je nyní velice obtížně identifikovatelné. Pokud ovšem ve snaze rozptýlit dávkové chyby původní větu "popřeházíme" na **TJoeřlPodlntríkPoaákáíd**, po čtyřech nastalých dávkových chybách dostaneme například **TJ_____lPodlntríkPoaákáíd**. Jednotlivá písmena nyní vrátíme na původní pozice, čímž získáme **Tot__J__P__íkladProk__ádání**. Dávkové chyby jsme tím rozptýlili mezi jednotlivá slova, která je mnohem lehčí přečíst. ■

Informace se zakóduje do RS-kódu, pak se provede prokládání a nakonec se ještě vše zakóduje do konvolučního kódu. Konvoluční kódy neoperují s bloky, ale s řetězci bitů, které se postupně zpracovávají. Na rozdíl od blokových kódů kódová slova konvolučních kódů nemusí mít pevně stanovenou délku (v případě Juno ale mají, protože do kodéru přichází bloky RS-kódu). Zpracovává se vždy postupně určitý počet bitů (nejčastěji 1) a na základě těchto bitů a určitého počtu již zpracovaných bitů se určí výstup. Na výstupu je větší počet bitů než byl na vstupu.

Konvoluční kódy jsou určené číslem K, které říká, že abychom určili výstup, potřebujeme také $K-1$ předchozích vstupů (tzn. pokud zpracováváme postupně po jednom bitu, tak potřebujeme právě zpracovávaný bit a dalších $K-1$ jeho předchůdců) a pak číslem R, které udává poměr mezi počtem vstupních a výstupních bitů. Konvoluční kód pak nazveme (K, R)-kód [7], [8].

Kód použitý v Juno je $(7, \frac{1}{2})$-kód, což znamená, že na základě vstupu jednoho bitu a jeho šesti předchůdců generujeme dvou bitový výstup [7].

Konvoluční kódy mají poměrně dobrou schopnost opravovat chyby, ale jednu zásadní nevýhodu. Když je dekodér konvolučního kódu přetížený, generuje dávkové chyby. Proto se používá kombinace RS-kódu a konvolučního kódu. Pořadí kódování a dekódování vypadá následovně [2]:



Obrázek 8: Pořadí kódování a dekódování pro kombinaci RS a konvolučního kódu

Situace je tedy taková, že s většinou samostatně stojících chyb, které při přenosu nastanou, si poradí konvoluční kód. Pokud nějaké chyby projdou (například zmíněné dávkové chyby), poradí si s nimi prokládaný RS-kód.

Kromě kombinace RS a konvolučního kódu používá pro zajištění bezchybného přenosu Juno ještě tzv. **turbo kód**. Turbo kódy jsou poměrně mladá skupina samoopravných kódů (první článek o turbo kódech pochází z roku 1993 [12]) a kromě komunikace skrz Vesmír se používají také v mobilních sítích LTE. Turbo kódy nejsou tolik náchylné na dávkové chyby jako konvoluční kódy a mohou proto stát samostatně.

Kódování turbo kódů funguje v podstatě na stejném principu jako kódování konvolučních kódů. Hlavní rozdíl je v dekódování, kdy se turbo kódy dekódují na principu dvou a více souběžně pracujících dekodérů. Každý dekodér vyřkne hypotézu a s ní spojenou věrohodnost (v podstatě jak moc si se svou hypotézou "věří"), následně své hypotézy porovnají a pokud se v odpovědi liší, vzájemně si vymění věrohodnosti pro každý dekódovaný bit, zakomponují je do nové hypotézy a dekódují znovu tak dlouho, dokud se neshodnou [2].

Turbo kód použitý v Juno kóduje v poměru $\frac{1}{6}$, tzn. že na každý bit na vstupu připadá šest bitů na výstupu [7].

4 Kód použitý v experimentu

V této kapitole detailně rozebereme kód, který jsme zvolili pro náš experiment.

Ačkoliv obvykle v praxi preferujeme co možná nejlepší schopnost opravovat chyby a důmyslné co nejefektivnější dekódovací algoritmy, v případě kódu pro experiment jsou naše preference v zásadě úplně opačné. Jelikož je experiment určen pro propagační účely a počítá se, že bude prezentován hlavně laikům, princip dekódování musí být snadno vysvětlitelný a také se nemusíme za každou cenu snažit opravit co nejvíce chyb, potřebujeme opravit alespoň nějaké. Proto nám stačí pouze malý obecný lineární kód.

Pokud tedy chceme opravit alespoň jednu libovolnou chybu ($t = 1$), potřebujeme kód s minimální vzdáleností d alespoň 3. Dále jsme si určili, že chceme, aby kód byl schopen kódovat čtyři různé zprávy 00,01,10 a 11, proto délka zprávy $k = 2$ a počet kódových slov $M = 4$. A protože kódová slova chceme co možná nejkratší, je potřeba nalézt co nejmenší n tak, aby existoval binární $[n, 2, 3]$ kód. Takové nejmenší n je $n = 5$.

Pro experiment tedy volíme binární lineární $[5, 2, 3]$-kód $C = \{00000, 01101, 10110, 11011\}$.

Generující matice G a kontrolní matice H tohoto kódu jsou:

$$G = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}, \qquad H = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Informační poměr tohoto kódu je $R(C) = \frac{k}{n} = \frac{2}{5}$. Za dobrý informační poměr se většinou považuje poměr větší než $\frac{1}{2}$ [3]. Z povahy experimentu se ovšem snažíme kód držet co možná nejkratší a kódovací a dekódovací algoritmy co možná nejjednodušší, hodnota dosaženého informačního poměru pro nás proto nehraje významnou roli. Pokud bychom chtěli dosáhnout lepšího informačního poměru, museli bychom kód zvětšit či přejít k již zmíněným důmyslnějším, ale zato z hlediska použitého matematického aparátu mnohem složitějším třídám kódů, jako jsou Reed-Solomonovy kódy.

4.1 Odhad počtu slov

Kolik nejvíce kódových slov bychom mohli použít, pokud chceme co nejkratší kód opravující jednu chybu? Dosaďme nyní parametry $n = 5$, $d = 3$ a $t = 1$ do Hammingova a Singletonova odhadu.

Z Hammingova odhadu dostáváme

$$M\left\{\binom{n}{0}+\binom{n}{1}(q-1)+\ldots+\binom{n}{t}(q-1)^t\right\}\leq q^n$$
$$M\left\{\binom{5}{0}+\binom{5}{1}(2-1)\right\}\leq 2^5$$
$$M\leq\frac{16}{3}.$$

Náš kód s konkrétním $M=4$ v odhadu nedosahuje rovnosti, není proto perfektní. Dále si můžeme všimnout, že jelikož M musí být celé číslo, tak $A_2(5,3)\leq 5$. Že dané parametry n,t a M splňují Hammingův odhad však ještě neznamená, že takový kód existuje. Pro $n=5$, $t=1$ a $M=5$ totiž neexistuje, což brzy ukážeme.

Ze Singletonova odhadu dostáváme

$$A_q(n,d)\leq q^{n-d+1}$$
$$A_2(5,3)\leq 2^3$$
$$A_2(5,3)\leq 8.$$

Ani zde náš kód nedosahuje rovnosti, není tudíž ani MDS. Zároveň vidíme, že Singletonův odhad by pro zvolené n a d dovolil dokonce 8 slov. Reálně jsou však 4 slova maximum.

Pokud by nás zajímal dolní odhad $A_2(5,3)$, po dosazení do Gilbert-Varshamova odhadu pro obecné kódy dostáváme

$$A_q(n,d)\geq\frac{q^n}{\binom{n}{0}+\binom{n}{1}(q-1)+\binom{n}{2}(q-1)^2+\ldots+\binom{n}{d-1}(q-1)^{d-1}}$$
$$A_2(5,3)\geq\frac{2^5}{1+\binom{5}{1}+\binom{5}{2}}$$
$$A_2(5,3)\geq 2.$$

Protože je náš kód lineární, můžeme parametry dosadit také do vztahu 13, čímž dostáváme

$$q^k<\frac{q^n}{\binom{n-1}{0}+\binom{n-1}{1}(q-1)+\binom{n-1}{2}(q-1)^2+\ldots+\binom{n-1}{d-2}(q-1)^{d-2}}$$
$$2^k<\frac{2^5}{1+4}$$
$$2^k<6,4.$$

Největší celé k takové, že splňuje předchozí nerovnost, je $k = 2$, a proto

$$A_q(n, d) \geq q^k$$
$$A_2(5, 3) \geq 2^2 = 4.$$

Z odhadů tedy dostáváme, že $4 \leq A_2(5, 3) \leq 5$. Ukažme nyní, že $A_2(5, 3) = 4$, a tudíž že pro námi zvolené parametry n a d nemůžeme sestrojit binární kód, který by měl více než $M = 4$ slova. Důkaz je kombinatorický.

Věta 15 *Maximální počet slov takový, že binární* $[5, k, 3]$*-kód existuje, je* 4*.*

Důkaz Předpokládejme, že máme binární $[5, k, 3]$-kód a $M > 4$. V prvé řadě máme lineární kód, proto jedno z kódových slov musí být nulový vektor 00000 (i kdybychom neměli lineární kód, každý kód je ekvivalentní s kódem o stejných parametrech, který obsahuje nulový vektor [3]). Tím pádem můžeme mít pouze jedno kódové slovo, které obsahuje více než 3 jedničky, protože kdyby byla taková slova dvě, měla by alespoň tři jedničky na stejné pozici, což by znamenalo, že $d \leq 2$. Stejně tak nemůžeme mít žádná kódová slova obsahující pouze dvě nebo jednu jedničku. Kromě jednoho nulového vektoru a jednoho slova se čtyřmi a více jedničkami proto můžeme mít už pouze slova se třemi jedničkami. Jelikož předpokládáme, že $M > 4$, musí být taková slova alespoň 3. Pro $n = 5$ ovšem není možné sestavit tři a více vektorů, které by obsahovaly právě 3 jedničky a přitom se vzájemně lišily na alespoň třech pozicích. Proto $A_2(5, 3) \leq 4$. A protože zcela jistě existuje $[5, k, 3]$-kód o čtyřech slovech (náš kód pro experiment), tak $A_2(5, 3) = 4$. ■

4.2 Dekódování

Dále si ukážeme dekódování pomocí zvoleného $[5, 2, 3]$-kódu. Standard array zvoleného kódu je

00000	**01101**	**10110**	**11011**
00001	01100	10111	11010
00010	01111	10100	11001
00100	01001	10010	11111
01000	00101	11110	10011
10000	11101	00110	01011
11000	10101	01110	00011
10001	11100	00111	01010

Tabulka 5: Standard array $[5, 2, 3]$-kódu

Jelikož je kód schopen opravit libovolnou jednu chybu, v prvním sloupci si můžeme všimnout, že skutečně obsahuje všechny chybové vzory váhy 1. Dále však obsahuje i dva vzory váhy 2, kód je proto schopen opravit i dvě specifické chyby. Tyto chyby musejí nastat na první a druhé nebo první a poslední pozici.

Nyní se budeme zabývat spolehlivostí dekódování. Pro uvedený standard array vypočteme pravděpodobnost, že správně dekódujeme na kódové slovo, jako

$$P_{corr}(C) = \sum_{i=0}^{n} \alpha_i p^i (1-p)^{n-i} = (1-p)^5 + 5p(1-p)^4 + 2p^2(1-p)^3,$$

pro konkrétní $p = 0,01$ je $P_{corr}(C) = 0,9992$.

Naproti tomu pravděpodobnost, že dekódujeme na špatné kódové slovo, je pro $p = 0,01$

$$P_{err}(C) = 1 - P_{corr}(C) = 1 - 0,9992 = 0,0008.$$

Jak jsme již nastínili, standard array máme možnost nahradit syndromovou tabulkou. Z povahy experimentu ovšem záměrně pracujeme s malým kódem, rozměry standard array proto nejsou nijak markantní a syndromová tabulka není potřeba. Zároveň je mnohem snazší laikům vysvětlit dekódovací proces hledáním v tabulce než násobením matic (experiment je určen hlavně žákům druhého stupně základních škol a středoškolákům, kteří pochopitelně nemusí mít nejmenší tušení, co jsou matice a jak s nimi zacházet). Pro úplnost a případné zájemce ovšem syndromovou tabulku přeci jenom uveďme.

Reprezentanti	Syndromy
00000	000
00001	001
00010	010
00100	100
01000	110
10000	101
11000	011
10001	111

Tabulka 6: Syndromová tabulka $[5, 2, 3]$-kódu

Při definování obecných blokových a lineárních kódů jsme uvažovali přenos skrze binární symetrický kanál. V něm se neuvažuje možnost, že symbol úplně vypadne, musí být nahrazen jiným symbolem. V experimentu ovšem chceme umět řešit i situace s vypadlými symboly. Pro obecné lineární kódy ale neexistuje žádný algoritmus, určený k řešení tohoto problému. Řešení nabízí až Reed-Solomonovy kódy, které při dekódování slov s vypadlými symboly využívají Euklidův algoritmus [2]. Z povahy experimentu jsme si nedovolili Reed-Solomonovy kódy používat, a proto jsme pro opravu vypadlých symbolů použili řešení popsané níže.

V experimentu (detailně popsaném později v kapitole 5.3) má uživatel plnou kontrolu nad nastalými chybami. Pokud tedy zajistíme, že vypadlý symbol bude jedinou chybou v právě odesílaném

kódovém slově a budeme přesně vědět, který v pořadí byl, můžeme jej jednoduše nahradit nulou. Pokud jsme se "strefili" a vypadlý symbol byl skutečně 0, zpráva dorazí v pořádku. Pokud jsme se "nestrefili" a symbol měl být 1, jelikož jsou všechny ostatní symboly správně, došlo k jedné chybě a zpráva bude opravena.

Příklad 18

Předpokládejme, že jsme odeslali kódové slovo 10110 a došlo k výpadku třetího symbolu. Vypadlý symbol označme X. Přijato bylo tudíž slovo 10X10. Vypadlý symbol nyní zkusíme nahradit symbolem 0, čímž dostáváme 10010. Toto slovo nekoresponduje s žádným kódovým slovem. Nalezneme jej ve standard array.

00000	**01101**	**10110**	**11011**
00001	01100	10111	11010
00010	01111	10100	11001
00100	01001	**10010**	11111
01000	00101	11110	10011
10000	11101	00110	01011
11000	10101	01110	00011
10001	11100	00111	01010

Pomocí standard array nyní správně opravíme na odeslané slovo 10110.

■

5 Popis experimentu

V této části detailně představíme a popíšeme experiment, jež je hlavní složkou této práce. Experiment má za cíl populárně naučnou formou demonstrovat přenos dat chybovým komunikačním kanálem a následnou opravu informace s využitím samoopravných kódů. Kromě popisu finální podoby experimentu a jeho funkčnosti popíšeme také použitou stavebnici a rozebereme možnosti jejího programování.

5.1 Stavebnice

Ke zkonstruování byla použita stavebnice **Lego Mindstorms**. Konkrétně se jedná o sady **31313 EV3** a **Education 45544 EV3**. Klíčovou komponentou obou těchto stavebnic je programovatelný řídící prvek (výrobcem označován jako *Inteligentní kostka*), který na základě uživatelských pokynů zajišťuje ovládání připojených periferií nebo periferií, které jsou jeho přímou součástí (reproduktor, display).

Obrázek 9: Inteligentní kostka

Stavebnice EV3 je třetí generací stavebnic Lego Mindstorms. Její dva předchůdci **RCX** a **NXT 1.0**/**NXT 2.0** byli plně nahrazeni a v běžných obchodech již nejsou k dostání. Řada EV3 oproti nim disponuje mikroprocesorem ARM9, který je schopen rozeběhnout operační systém Linux, dále USB konektorem a slotem pro MicroSD paměťovou kartu.

Základní sada 31313 EV3 kromě inteligentní kostky, kabelů a stavebních lego kostiček obsahuje 2 velké motory, 1 střední motor, 2 dotykové senzory, 1 barevný senzor, 1 infračervený senzor a dálkový ovladač.

Education sada má oproti základní sadě 1 dotykový senzor navíc a namísto infračerveného senzoru s dálkovým ovládáním obsahuje 1 gyroskopický a 1 ultrasonický senzor.

Pro náš experiment byl klíčový hlavně barevný senzor, operující ve dvou režimech. Senzor barev, který je schopen na krátkou vzdálenost určovat barvu objektů a senzor jasu, vracející číselnou hodnotu reprezentující intenzitu odraženého světla. Dále byly využity také motory a dotykový senzor, který lze nastavit, aby reagoval na stisknutí či puštění (návrat do původní polohy).

Obrázek 10: Barevný senzor

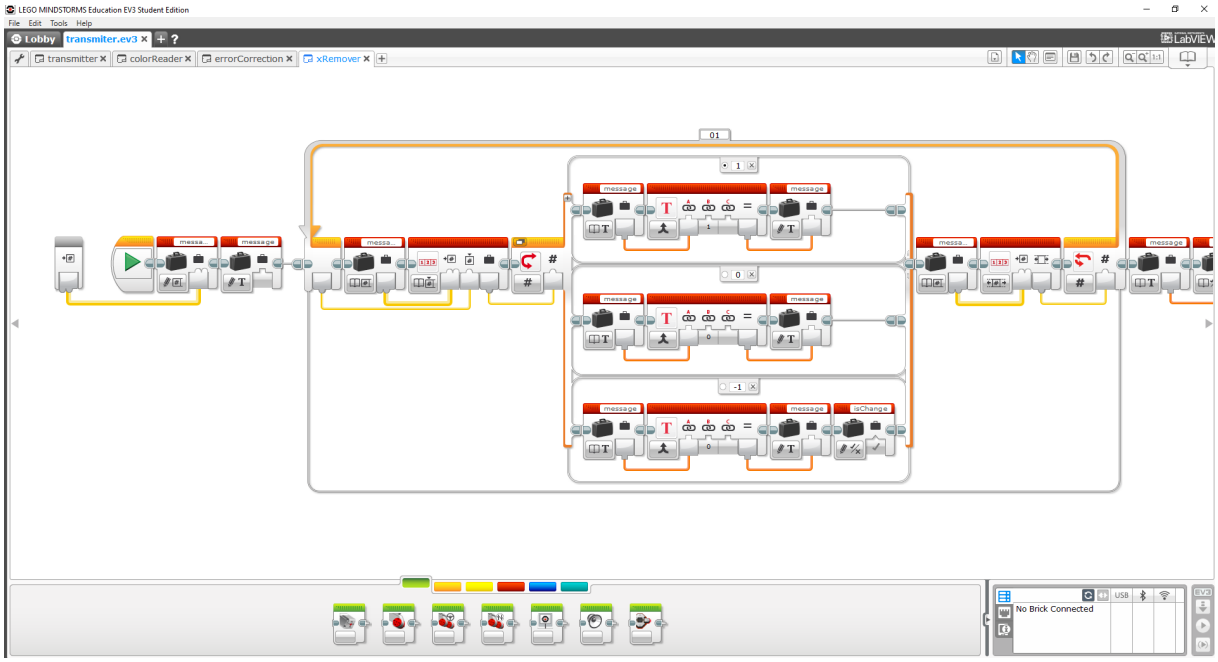
Obrázek 11: Dotykový senzor

5.2 Možnosti ovládání

Stavebnice Lego Mindstorms není omezena pouze na jeden způsob ovládání. Robota, zkonstruovaného pomocí přiložených návodů, dokonce není ani nutno ručně programovat. Uživatel si totiž může zdarma stáhnout před připravenou aplikaci (ať už do počítače nebo mobilního telefonu), pomocí které do robota nahraje již hotový program (přes USB či Bluetooth), případně lze aplikaci využít pouze jako dálkový ovladač a předávat přímé pokyny.

Pokud ovšem chceme robotovi předávat vlastní složitější instrukce, musíme jej patřičně naprogramovat. Výchozí volbou je software **SW LME EV3** (stažitelný z: [15]). Jeho vývojové prostředí vychází z programovacího jazyka **LabVIEW**, jedná se tedy o tzv. *Vizuální progra-*

movací jazyk. Software je zcela zdarma a umožňuje využívat veškeré funkce, které stavebnice nabízí. Samozřejmě ale existují i další vizuální programovací jazyky, využitelné se stavebnicí.



Obrázek 12: Vývojové prostředí SW LME EV3

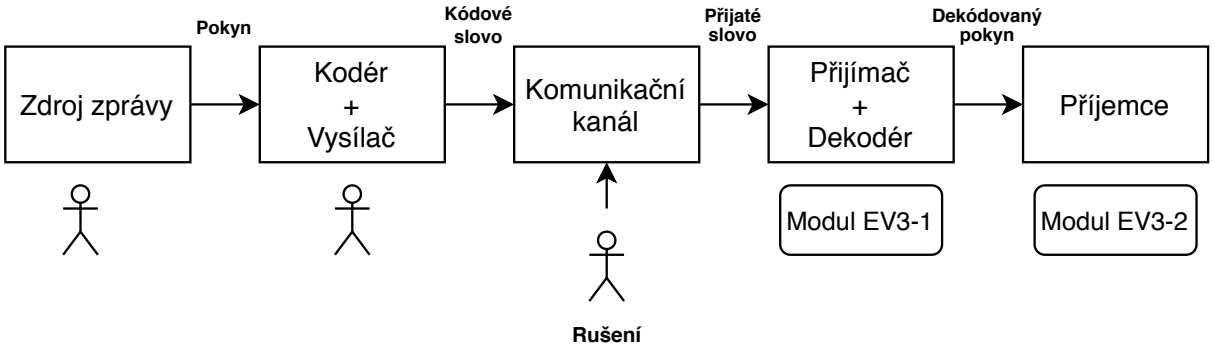
Nechceme-li z nějakého důvodu používat vizuální programování, lze sáhnout také po "tradičnějších" jazycích. Použít můžeme například programovací jazyk **RobotC** (oficiální stránky: [17]), který, jak už název napovídá, vychází svou strukturou z jazyka C. Jazyk disponuje přívětivým uživatelským prostředím a poměrně solidními nástroji k ladění kódu. Zároveň lze spolu s jazykem využít tzv. *Robot Virtual Worlds*, kdy můžeme napsaný kód otestovat na virtuálním robotovi (v podstatě není třeba ani vlastnit stavebnici). K využívání tohoto jazyka je ovšem nutno zakoupit licenci. Další možností k programování pak může být také software **MATLAB** (nutno doinstalovat toolbox dostupný z: [16]).

5.3 Experiment

V této části se budeme věnovat samotnému detailnímu popisu experimentu. Jak už bylo zmíněno, experiment má sloužit jako názorná pomůcka k demonstraci fungování samoopravných kódů. Proto je chyba, kterou je nutno opravit, realizována nezvykle nikoliv při přenosu, nýbrž ještě před odesláním. Chyby má plně pod kontrolou uživatel.

Experiment se skládá ze dvou hlavních prvků (dvou robotů), které pro přehlednost označme **modul EV3-1** a **modul EV3-2**.

Schéma experimentu je následující:



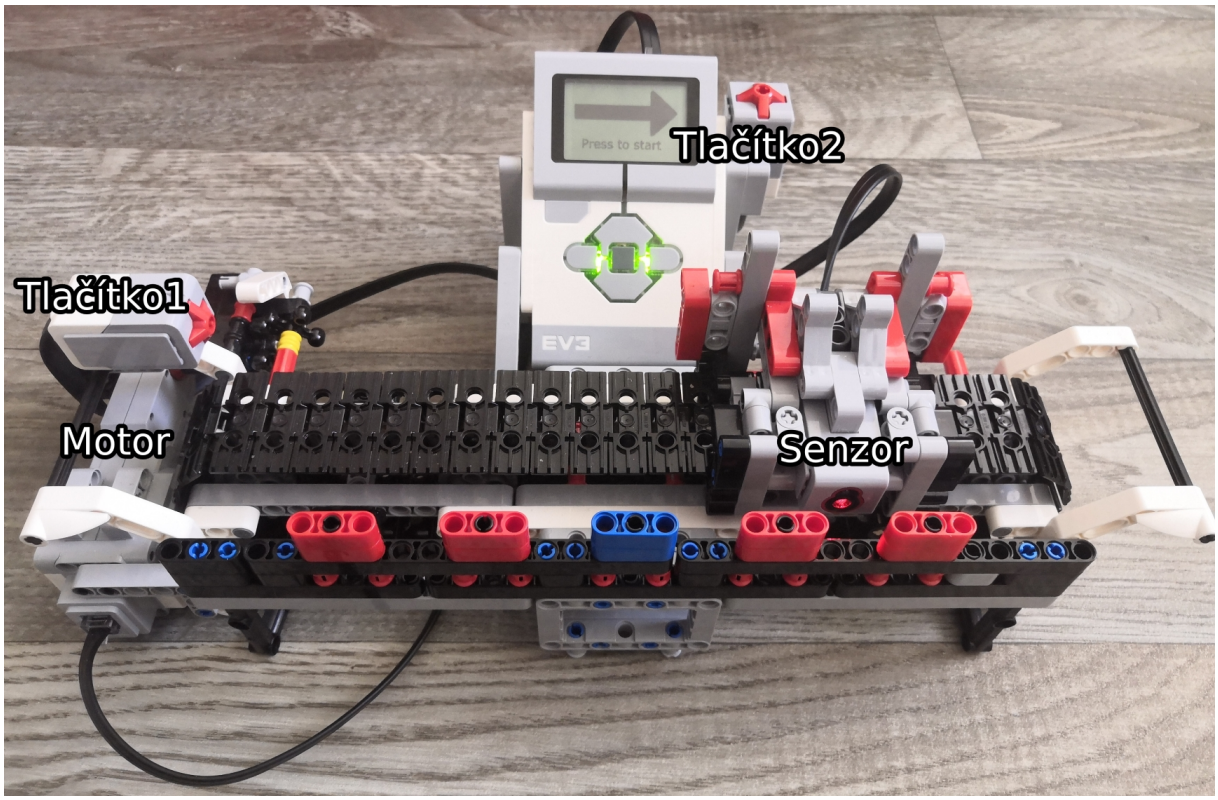
Obrázek 13: Schéma experimentu

V následujících podkapitolách podrobně rozebereme jednotlivé moduly.

5.3.1 Modul EV3-1

Ke konstrukci prvního modulu byla využita hlavně Education sada, přičemž z důvodu nedostatku Lego stavebních kostek bylo potřeba jich pár přidat i z druhé dostupné sady. Modul slouží jako čtečka zpráv, které si uživatel přeje odesílat. Princip čtení informace je založen na barvách, kdy je binární zpráva reprezentována barevnými kostičkami vyskládanými za sebou na před připravená místa. Jednotlivé barvy reprezentují kódové symboly (červená - 1, modrá - 0). Senzor schopný rozpoznávat barvy, připevněný na pohyblivém páse, poté jednoduše projede kolem kostek a interpretuje zprávu. Chyba při přenosu je realizována záměnou správné kostky za kostku s jinou barvou, případně lze kostku úplně zakrýt a znemožnit tak senzoru přečtení daného symbolu.

Následující obrázek ukazuje výslednou podobu čtečky. Dále je uveden podrobný popis funkcionality.



Obrázek 14: Modul EV3-1

Po spuštění program čeká, až uživatel stiskne dotykový senzor (v obrázku označen jako *Tlačítko2*). Stisk spustí inicializační fázi, kdy se barevný senzor (ozn. *Senzor*) automaticky posouvá po páse směrem vlevo. Posun je poháněn motorem (ozn. *Motor*) umístěným vedle pásu. K zastavení dojde v momentě, kdy barevný senzor narazí na dotykový senzor (*Tlačítko1*). Tímto způsobem je zaručeno, že barevný senzor bude po inicializaci vždy na stejném místě bez ohledu na jeho předchozí polohu na páse.

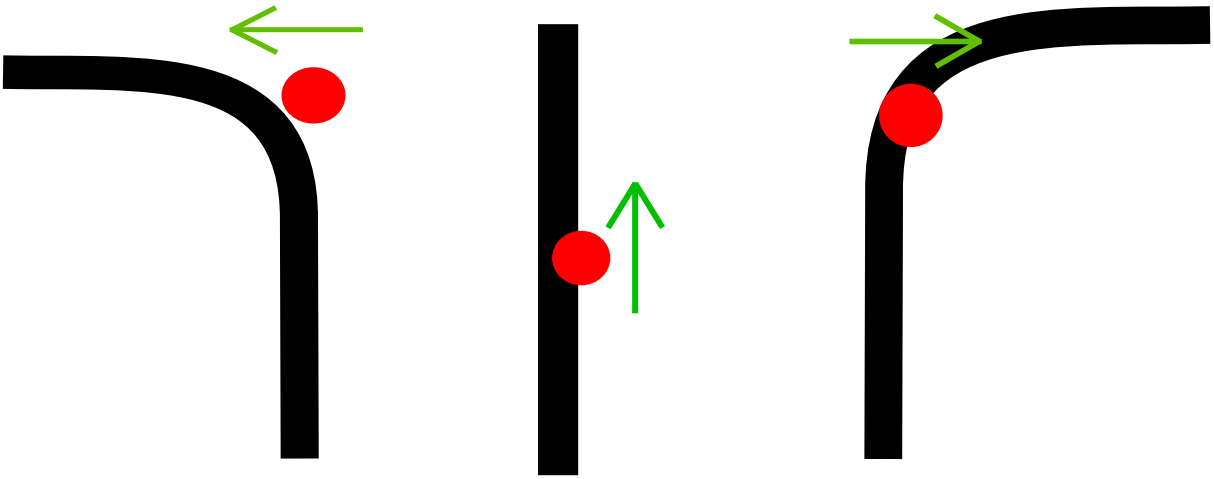
Po inicializaci program vyzve uživatele, aby si zvolil, zda chce použít funkci opravování chyb. Možnost volby je zde z toho důvodu, aby bylo možno demonstrovat případné odeslání neopravené zprávy. Následně robot vyčkává na pokyn k přečtení a odeslání zprávy (*Tlačítko2*).

Jakmile uživatel stiskem potvrdí výzvu pro čtení, začne *Motor* posouvat pás a s ním i *Senzor*, přičemž postupně dochází k předem definovaným zastávkám na místech, kde se nachází barevné Lego kostičky reprezentující symboly 0 a 1. Jakmile dorazí k poslední kostičce, na základě předchozí uživatelské volby buď zprávu zkontroluje a opraví, či nechá být. Kompletní zprávu následně odešle pomocí Bluetooth do modulu EV3-2. Při tomto přenosu již k chybám nedochází, jelikož

bluetooth komunikace je uvnitř modulů realizována na bázi “zpráva buď přijde bez chyb nebo vůbec (dojde ke ztrátě spojení)”.

5.3.2 Modul EV3-2

Jelikož je veškeré opravování chyb realizováno již v EV3-1, je role modulu EV3-2 poměrně přímočará. Jeho úkolem je reagovat na přijatá slova pomocí předem definovaných pokynů. A protože je experiment určen hlavně k propagačním účelům (a měl by tedy na sebe přivádět pozornost), důraz byl kladen také na to, aby byl modul EV3-2 neustále v pohybu i tehdy, když pouze vyčkává na další zprávu. K sestavení byla využita sada 31313 EV3. Důležitou součástí, stejně jako u EV3-1, je barevný senzor, který je zde využit za účelem následování dráhy. Senzor nezkoumá pouze barvu, nýbrž intenzitu odraženého světla. Následování dráhy je pak založeno na jednoduchém principu, kdy černá barva světlo pohlcuje a bílá odráží. Pokud je hodnota odraženého jasu cca na padesáti procentech, senzor vidí rozhraní mezi černou dráhou a bílým podkladem, robot v tomto okamžiku směřuje podél dráhy a jede rovně. Pokud je hodnota na nula či sto procentech, senzor vidí pouze dráhu či podklad a robot musí zatočit odpovídajícím směrem, aby dráhu neopustil.



Obrázek 15: Princip následování dráhy

Modul EV3-2 se pohybuje pomocí dvou motorů, které pohánějí “tankové pásy”. Bylo proto nutné sestavit funkci, která na základě snímaných jasových hodnot v rozmezí 0 až 100 % přiřazuje rychlost jednotlivým motorům. Nastavení rychlosti motorů funguje na principu pulzně šířkové modulace. Číselná hodnota, která se u motorů volí, reprezentuje tzv. power level (pl). Pokud hodnotu nastavíme například na 25 pl, ze 100 cyklicky se opakujících pulzů bude 25 pulzů “zapnutých” a 75 “vypnutých”. Čím více pulzů je zapnutých, tím rychleji se motor otáčí. Záporná hodnota znamená otáčení opačným směrem. Experimentálně bylo zvoleno, že pokud má robot jet rovně (hodnota jasu je kolem 50 %), požadujeme, aby se rychlost obou motorů nastavila na 30 pl. Pokud má zatáčet (hodnota je přibližně 100 % nebo 0 % v závislosti na směru jízdy a zatá-

čení), jeden z motorů se nastaví na 85 pl a druhý na -20 pl. Dále bylo zjištěno, že nejplynulejšího pohybu dosahuje modul při použití lineární funkce (využití funkcí vyšších řádů mělo za následek trhavý pohyb a robot často sjížděl mimo dráhu). A protože zvolených hodnot nepotřebujeme dosáhnout úplně přesně (nezáleží, jestli je rychlost motoru 30 nebo 31 pl), k nalezení příslušných funkcí určujících rychlost motorů jsme použili **metodu nejmenších čtverců**. Demonstrujme nyní výpočet.

Označme hodnotu jasu x. Pak rychlost motoru $f(x) = ax + b$, kde $a, b \in \mathbb{R}$. Koeficienty a a b vypočteme následovně:

$$a = \frac{n\sum XY - \sum X \sum Y}{n\sum X^2 - (\sum X)^2},\ b = \frac{\sum Y - a\sum X}{n},$$

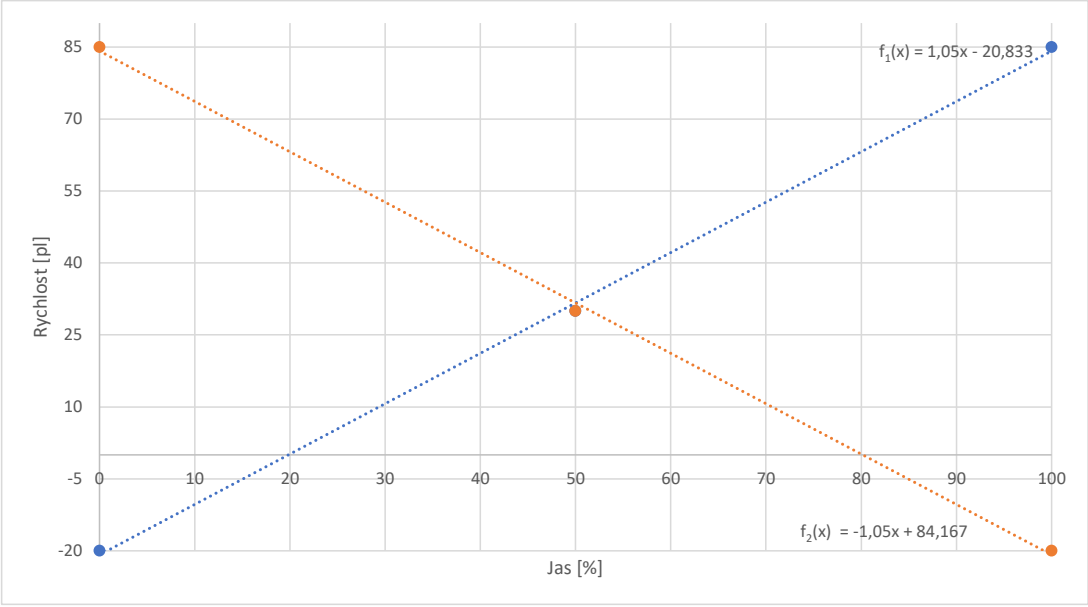
kde n je počet bodů v rovině (o souřadnicích X a Y) jimiž prokládáme přímku.

Pokud tedy dosadíme námi zvolené tři body $[100, 85]$,$[50, 30]$ a $[0, -20]$, dostaneme:

$$a = \frac{3 \cdot 10000 - 14250}{3 \cdot 12500 - 22500} = 1,05,\ b = \frac{95 - 1.05 \cdot 150}{3} = -20,8.$$

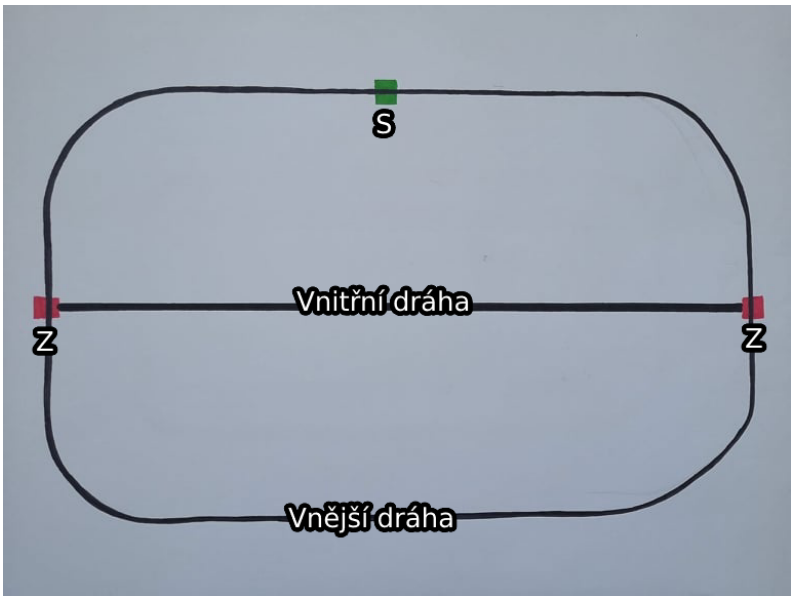
Obdobně vypočteme i koeficienty funkce pro druhý motor s požadovanými body $[100, -20]$,$[50, 30]$ a $[0, 85]$.

Výsledné funkce použité v experimentu, které určují rychlost motoru na základě jasu, jsou $f_1(x) = 1,05x - 20,8$ a $f_2(x) = -1,05x + 84$. Jejich grafy vypadají takto:



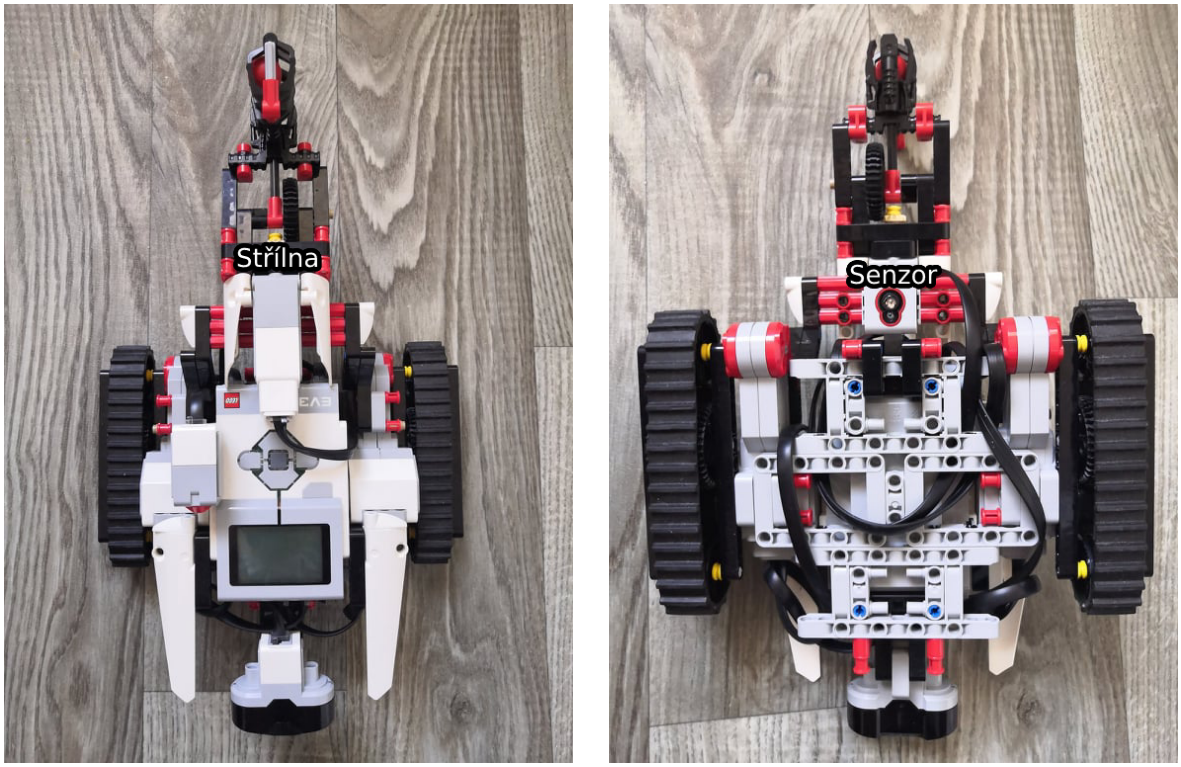
Obrázek 16: Funkce určující rychlost motorů

Dráha pro experiment byla vyrobena jednoduše s využitím barevných fixů a papíru o velikosti A1. Vypadá následovně:



Obrázek 17: Dráha

Samotný modul pak vypadá takto:



Obrázek 18: Modul EV3-2

Po spuštění programu robot jede po oválu (v obrázku označeném jako *Vnější dráha*) směrem tak, aby v zatáčkách zatáčel doprava. K následování čáry využívá, jak již bylo popsáno výše, barevného senzoru (ozn. *Senzor*), umístěného zespodu. Takto krouží, dokud nedostane zprávu korespondující s jedním ze čtyř možných pokynů. Pokyny jsou:

1. 

- Změna směru jízdy

2. 

- Vjezd na vnitřní dráhu

3. 

- Vjezd na vnější dráhu

4. 

- Střelba

Pokud robot obdrží pokyn 1 a nachází se na vnější dráze, ihned po přijetí pokynu, bez ohledu na to, kde se zrovna v danou chvíli nachází, učiní otočku o 180 stupňů a pokračuje v jízdě dokud nedostane další pokyn. Pokud se pohybuje po vnitřní dráze, kde otáčení nemá příliš smysl, z reproduktoru se ozve omluvná zpráva "sorry" a robot opět pokračuje v jízdě.

Po obdržení pokynu 2 robot začne kromě následování čáry hledat na dráze také jednu ze dvou červených značek (ozn. Z), které signalizují místo, kde má robot sjet z dráhy a zatočit na dráhu druhou. Hledáním červené značky dojde k poměrně velkému snížení přesnosti jasového senzoru. Je proto nutné, aby robot ve "vyhledávacím módu" zpomalil (jinak by mohl sjet z dráhy nebo značku úplně minout). Jednotlivé funkce, přiřazující rychlost motorům, se v tuto chvíli zamění za $f_1'(x) = 0,6x - 10$ a $f_2'(x) = -0,6x + 50$. Jakmile je změna dráhy u konce, robot jede po vnitřní dráze a pokaždé, když dojede k červené značce, se otočí o 180 stupňů a takto pokračuje, dokud nedostane další pokyn. Pokud robot přijme pokyn 2 a na vnitřní dráze se již nachází, opět se z reproduktoru ozve omluvná zpráva "sorry".

Pokyn 3 funguje principiálně úplně stejně, jako pokyn 2.

Po přijetí pokynu 4 robot začne za jízdy vyhledávat zelenou značku (ozn S). Jakmile ji nalezne, otočí se o 90 stupňů směrem dovnitř do dráhy, za pomoci připevněného motoru a mechanismu (ozn. *Střílna*) vystřelí kuličku, otočí se zpět a pokračuje v jízdě, dokud nedostane další pokyn. Pokud se po přijetí pokynu 4 robot nachází na vnitřní dráze, nejprve přejede na vnější dráhu a pak pokračuje stejným postupem.

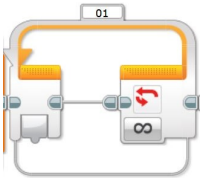
Pokud robot v důsledku vypnutí opravování chyb obdrží řetězec, který nekoresponduje s žádným ze čtyř pokynů, zmateně se dvakrát otočí tam a zpět, z reproduktoru se ozve chybová hláška “error” a robot pokračuje ve vykonávání pokynu, který prováděl, než zprávu obdržel.

5.4 Zdrojový kód

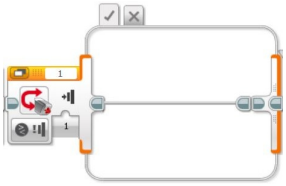
V této podkapitole uvedeme a stručně okomentujeme ukázku zdrojových kódů některých funkcí. Nebudeme uvádět celý kód, protože vizuální programovací jazyk, použitý v experimentu, se na rozdíl od klasických programovacích jazyků neformátuje shora dolů, ale zleva doprava, není proto možné jej přehledně a srozumitelně zobrazit na stránku.

5.4.1 Základní bloky

Než předvedeme samotnou ukázku, rozebereme nejdříve některé základní programovací bloky, aby bylo možno zdrojový kód lépe číst a pochopit (oficiální dokumentace: [14]).



Standardní cyklus, vpravo lze nastavit libovolnou ukončovací podmínku.



Switch blok, na základě vstupu se přejde do dané větve kódu.

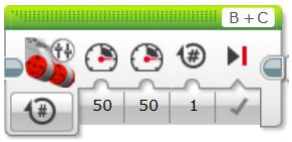
Pozastavení programu po daný čas.

Vynutí ukončení cyklu.

Vykreslí text nebo obrázek na display.

Rozsvítí nebo rozbliká hlavní kontrolku na inteligentní kostce jednou ze tří možných barev (červená, žlutá, zelená).

Blok pro nastavení rychlosti motoru.

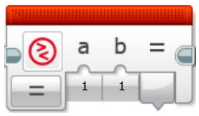


Nastavení rychlosti dvěma motorům současně (pro tankové pásy).

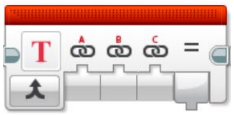
Reproduktor inteligentní kostky přehraje vybraný zvukový soubor se zvolenou hlasitostí.

Blok pro práci s proměnnými, umožňuje čtení a zápis.

Blok pro práci s poli.



Porovnávání číselných proměnných (větší, menší, rovno).



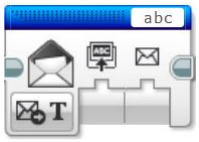
Konkatenace řetězců.

Blok pro provádění matematických operací.

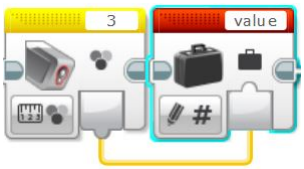
Vrací hodnotu nasnímanou senzorem barev.

Vrací hodnotu nasnímanou dotykovým senzorem.

Blok sloužící k navázání bluetooth spojení.



Odesílání a příjem zpráv skrze bluetooth.

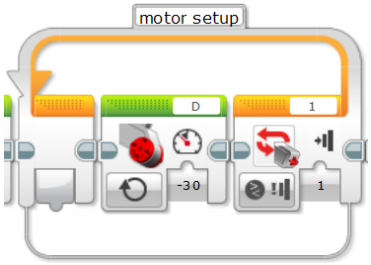


Ukázka řetězení bloků, hodnota nasnímaná senzorem barev se zapíše do proměnné “value”.

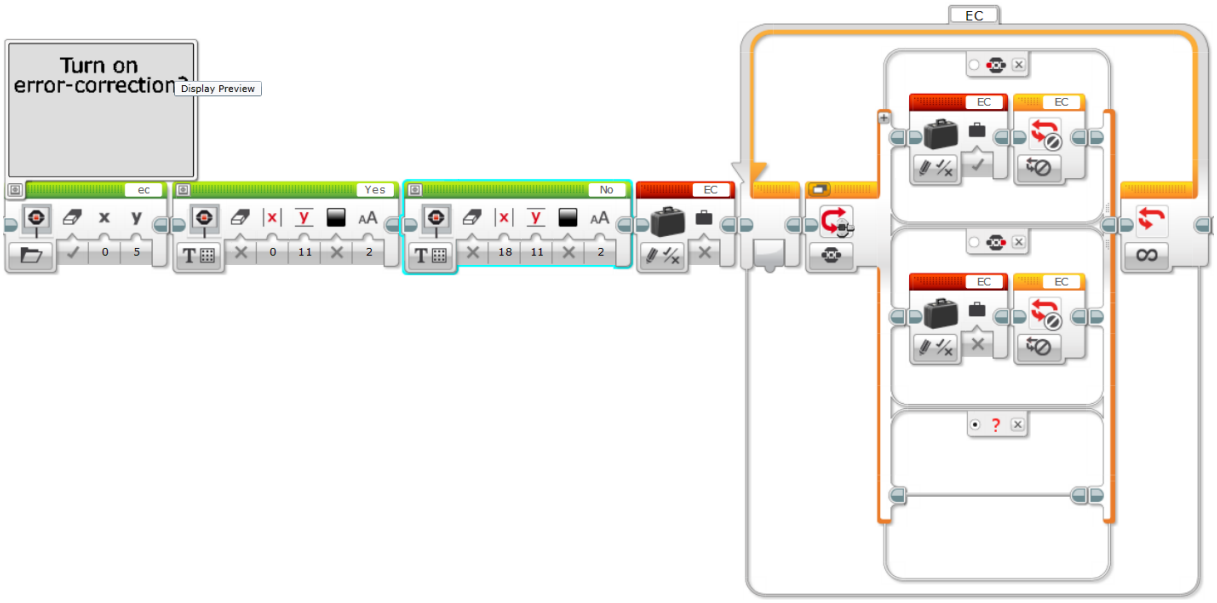
5.4.2 Modul EV3-1

Modul EV3-1 slouží k interpretaci a odeslání zprávy. Nejprve uvedeme ukázky z hlavní funkce.

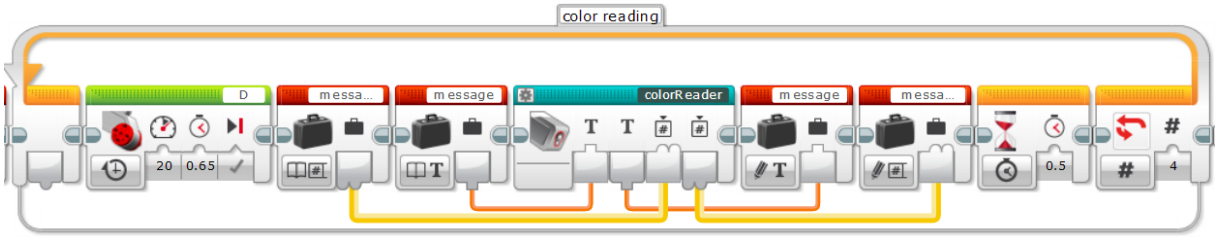
První ukázka uvádí inicializaci čtečky. Motor posouvá čtečku po páse směrem vlevo dokud nenarazí na dotykový senzor.



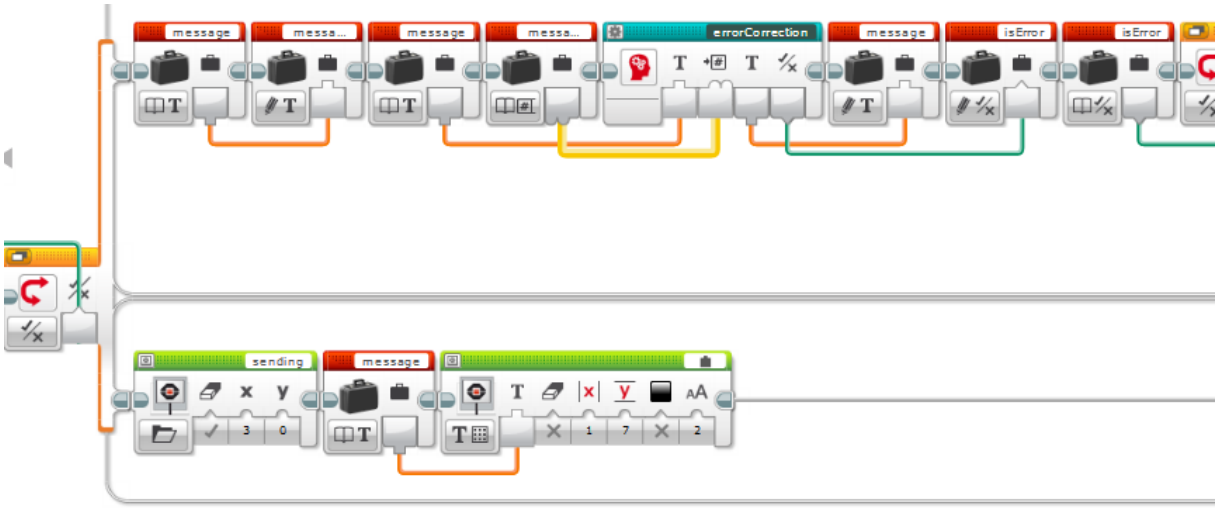
Na druhé ukázce nejprve dojde k výpisu na display a následně se čeká, až si uživatel zvolí, zda chce zapnout korekci chyb či nikoliv.



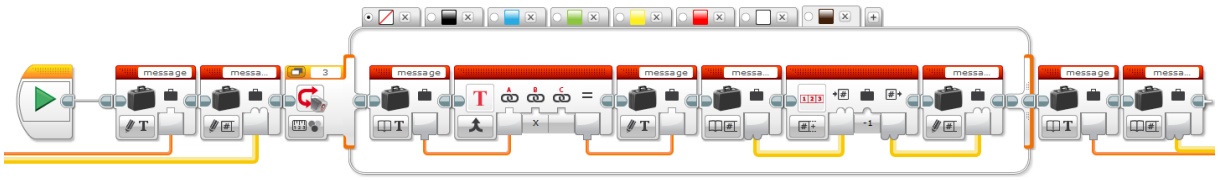
Dále můžeme vidět interpretaci zprávy. V cyklu se čtečka nejprve posune na místo, kde se nachází barevná kostička, následně se zavolá funkce **colorReader** a hodnota se přidá do proměnné se zprávou.



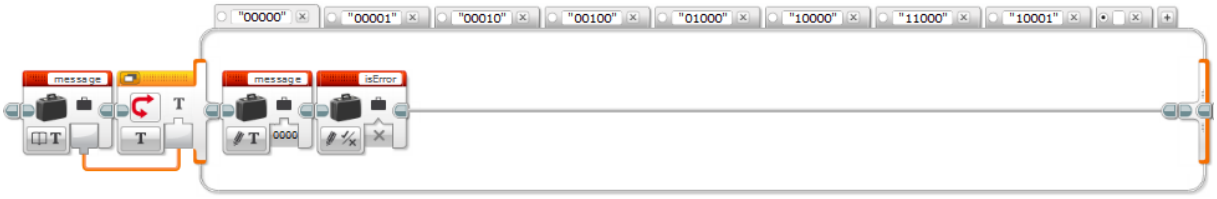
Poslední ukázkou z hlavní funkce je korekce chyb. Na základě uživatelské volby se buď bude zpráva opravovat zavoláním funkce **errorCorrection** (horní část) nebo se bude rovnou odesílat (dolní část).



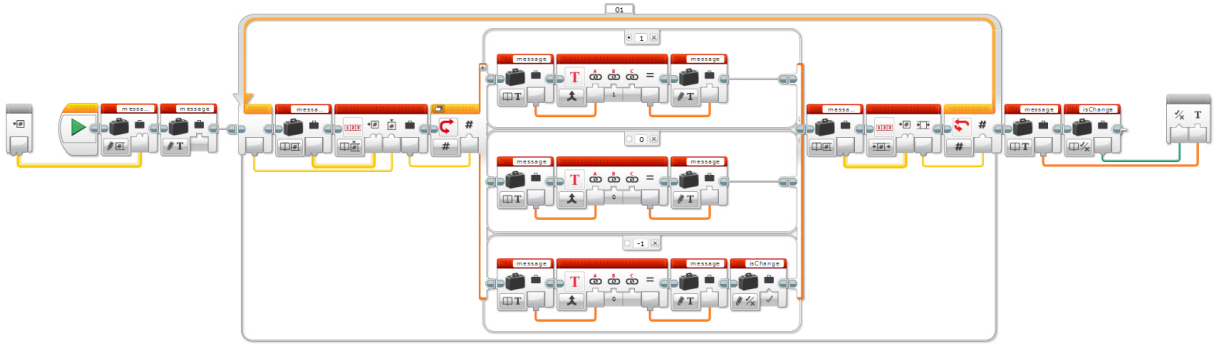
Nyní uveďme ukázky pomocných funkcí. První z nich je funkce **colorReader**, která na základě hodnoty barevného senzoru přidá do proměnné se zprávou buď 0,1 nebo X pro nečitelný symbol.



Další je ukázka z funkce **errorCorrection**. Zde probíhá dekódování pomocí standard array, tzn. vyhledáváním v tabulce.



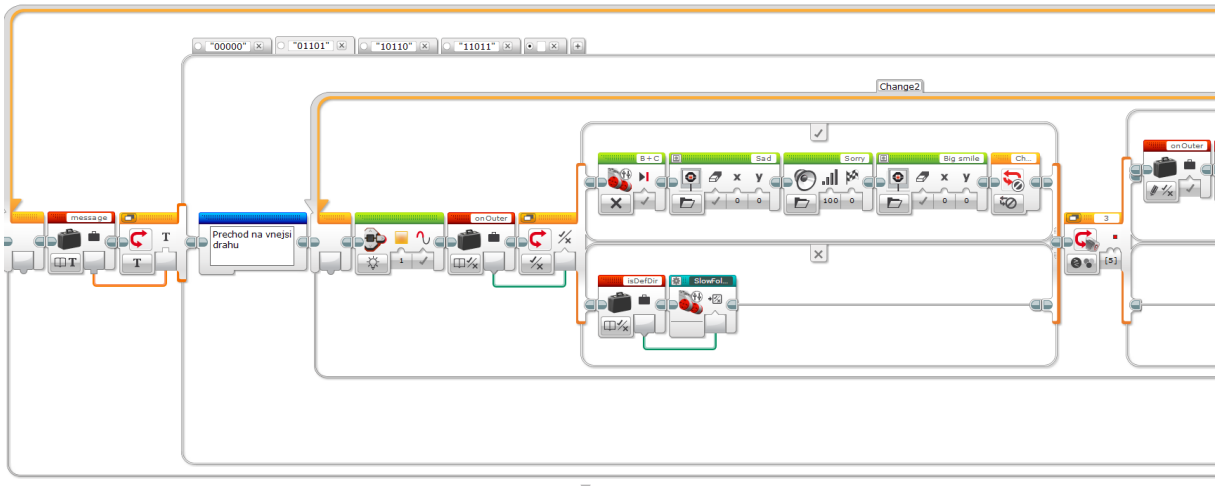
Poslední pomocnou funkcí je **xRemover**. Funkce se volá před zahájením opravy chyb. Všechny symboly X jsou nahrazeny symbolem 0.



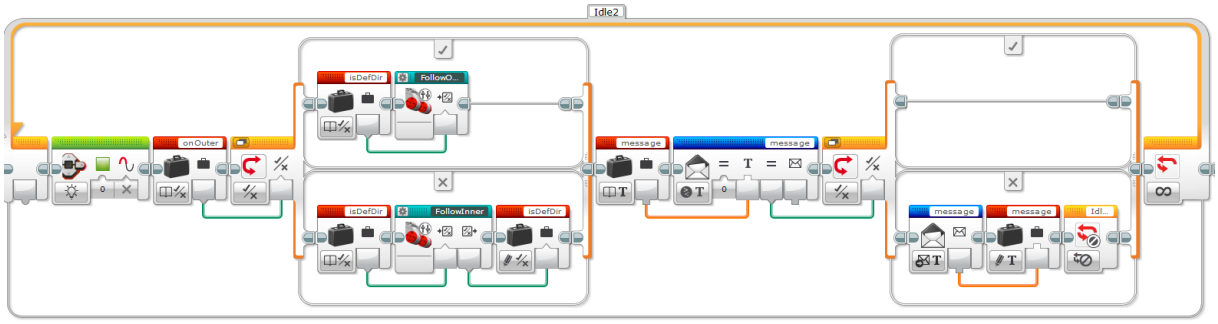
5.4.3 Modul EV3-2

Modul EV3-2 přijímá zprávy z modulu EV3-1 a reaguje odpovídajícím způsobem. Opět nejdříve uvedeme ukázky hlavní funkce.

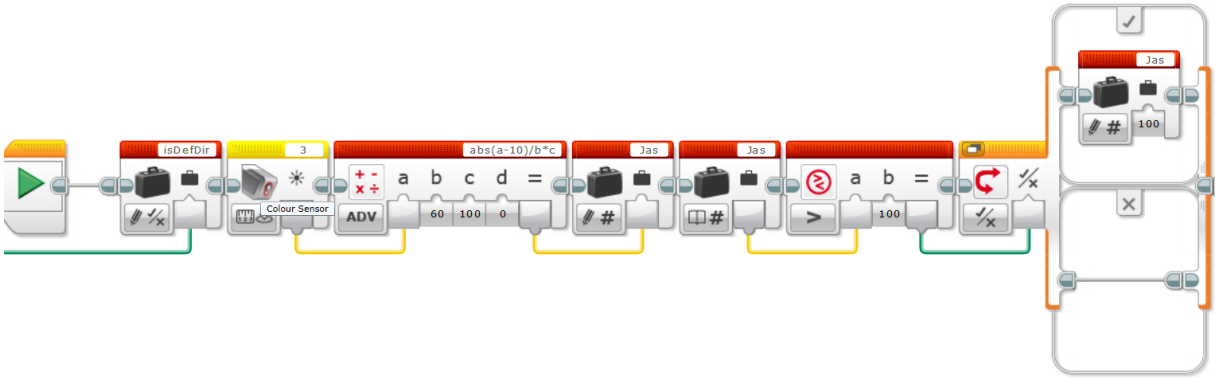
Na první ukázce lze vidět hlavní cyklus, kde se na základě přijaté zprávy volají pokyny.



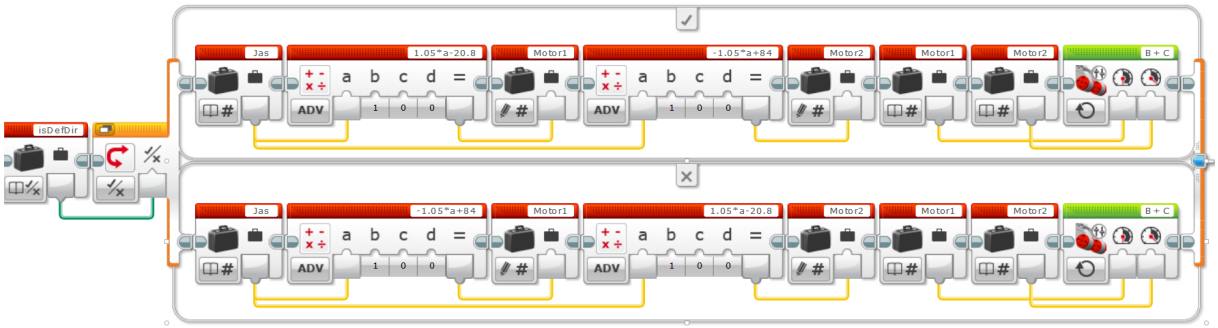
Na druhé ukázce vidíme čekání na zprávu, tzn. robot jezdí na jedné ze dvou drah dokud nedostane novou zprávu (pokyn).



První pomocnou funkcí je **followOuter**, sloužící k následování vnější dráhy. Z této funkce ukážeme dvě ukázky, nejprve jde o normalizaci nasnímané hodnoty jasu do intervalu $\langle 0, 100 \rangle$.

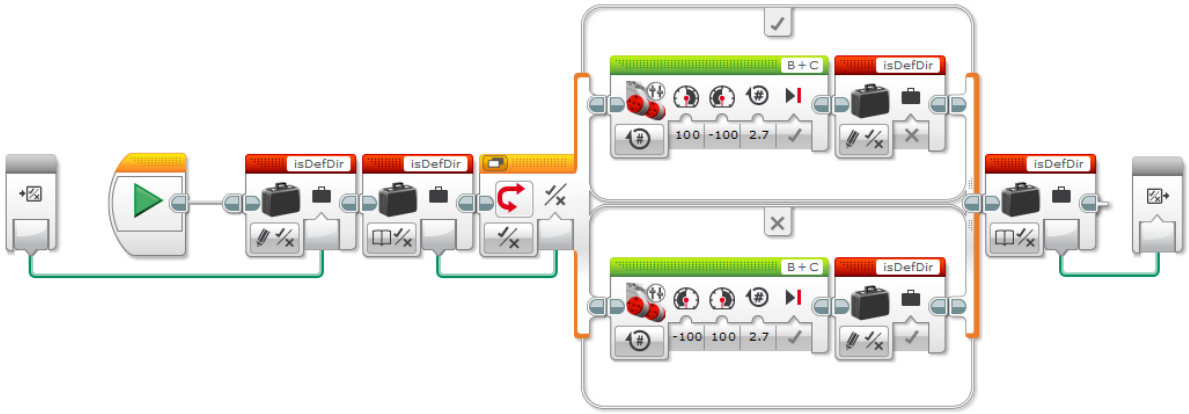


Poté na základě této hodnoty jasu přiřadíme hodnotu rychlosti oběma motorům.

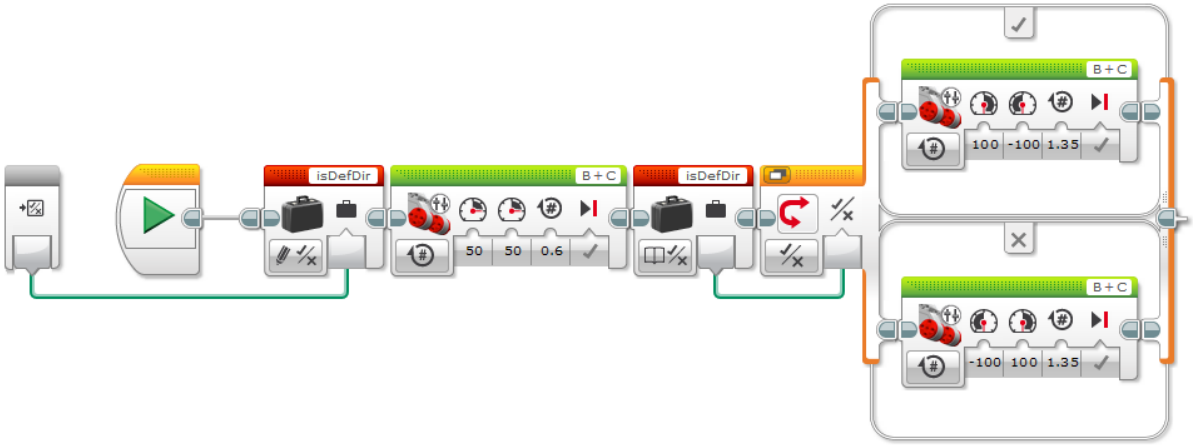


Funkci **followInner**, sloužící k následování vnitřní dráhy, není důvod uvádět, principem je velmi podobná funkci **followOuter** s tím rozdílem, že funkce přiřazující rychlosti motorům jsou jiné a pokaždé, když senzor barev narazí na červenou značku, se zavolá funkce **changeDir**.

Funkce **changeDir** slouží k otočení o 180 stupňů.



Poslední ukázkou je funkce **turn**, která se volá, když robot přejíždí z jedné dráhy na druhou.



5.5 Závěrečné poznámky k experimentu

Volba stavebnice a problémy s ní spojené do značné míry ovlivnily výslednou podobu experimentu. Zvolit si Lego se ukázalo jako dobrá volba. Stavební prvky jsou kvalitní, v jednotlivých sadách je mnoho periferií, které spolehlivě fungují a celkově práce s Lego Mindstorms je jednoduchá a přitom nabízí spoustu různých možností využití. Zároveň přesně plní stěžejní požadavek kladený na experiment, tj. přitahuje pozornost. Lidé obecně Lego dobře znají, je oblíbené, dostupné i běžnému zákazníkovi a pracovat s ním zvládnou bez obtíží i děti, kterým je primárně experiment určen.

Co se týče problémů, jedním z prvotních úskalí bylo vůbec vymyslet, jak bude probíhat simulace nastalých chyb. Protože je experiment určen hlavně k propagaci a předpokládá se, že bude prezentován laikovi převážně ve věku 10 až 18 let, nemohli jsme si jednoduše říci, že chyby

budeme simulovat uvnitř (třeba náhodným generováním). Bylo potřeba vymyslet způsob, který bude zřejmý a každému bude jasné, že skutečně k chybám dochází a se zprávou zjevně není něco v pořádku.

Chyby taktéž nebylo možné simulovat při přenosu zprávy pomocí Bluetooth (např. odstíněním nebo přesáhnutím maximální dovolené vzdálenosti), protože komunikace mezi dvěma inteligentními kostkami, jak už bylo zmíněno, probíhá zcela bez chyb nebo neprobíhá vůbec.

Klíčová vlastnost, schopnost posílat skrze Bluetooth zprávu z jedné inteligentní kostky do druhé, se nakonec ukázala jako největší problém ze všech. Původně totiž bylo zamýšleno, že k programování bude využit programovací jazyk RobotC (oficiální stránky: [17], dokumentace: [18]), jelikož se jevil jako nejvhodnější kandidát (v příloze A je zdrojový kód pro vysílač). Ke dni, kdy tato práce vzniká, ovšem jazyk stále nemá implementovánu právě požadovanou podporu Bluetooth komunikace, bylo proto nutné hledat alternativu. Jako vhodný nástupce se posléze jevil MATLAB. Dohledány byly dva způsoby, jak s pomocí něj roboty programovat.

První z nich je snadno stažitelný oficiální toolbox [16], který bohužel taktéž nemá podporu pro Bluetooth. Druhý je tzv. QUT toolkit, který ve své dokumentaci uvádí, že podporu implementovánu má. Zároveň je zde však uvedeno, že spojení ne vždy lze navázat, což bylo experimentálně potvrzeno.

Z toho důvodu je veškeré programování v našem experimentu realizováno implicitním vizuálním programovacím jazykem, který je velmi jednoduchý na užívání, ale nedovoluje složitější konstrukce a limituje tak celý experiment (např. implementaci složitějších kódovacích a dekódovacích algoritmů).

Zároveň zmiňme, že existuje i možnost s pomocí microSD karty do robota nahrát jiný operační systém, což umožní programování například jazykem Python nebo Java. Tato možnost ovšem nebyla převážně z časových důvodů odzkoušena. Jelikož se nepodařilo dohledat potvrzení o funkčnosti Bluetooth komunikace mezi EV3 kostkami s použitím těchto jazyků, byla zvolena možnost, která sice limituje, ale s jistotou splňuje všechny klíčové požadavky kladené na experiment a je snadno dostupná všem potenciálním zájemcům, kteří by chtěli tento experiment reprodukovat.

5.6 Rekapitulace funkcionality

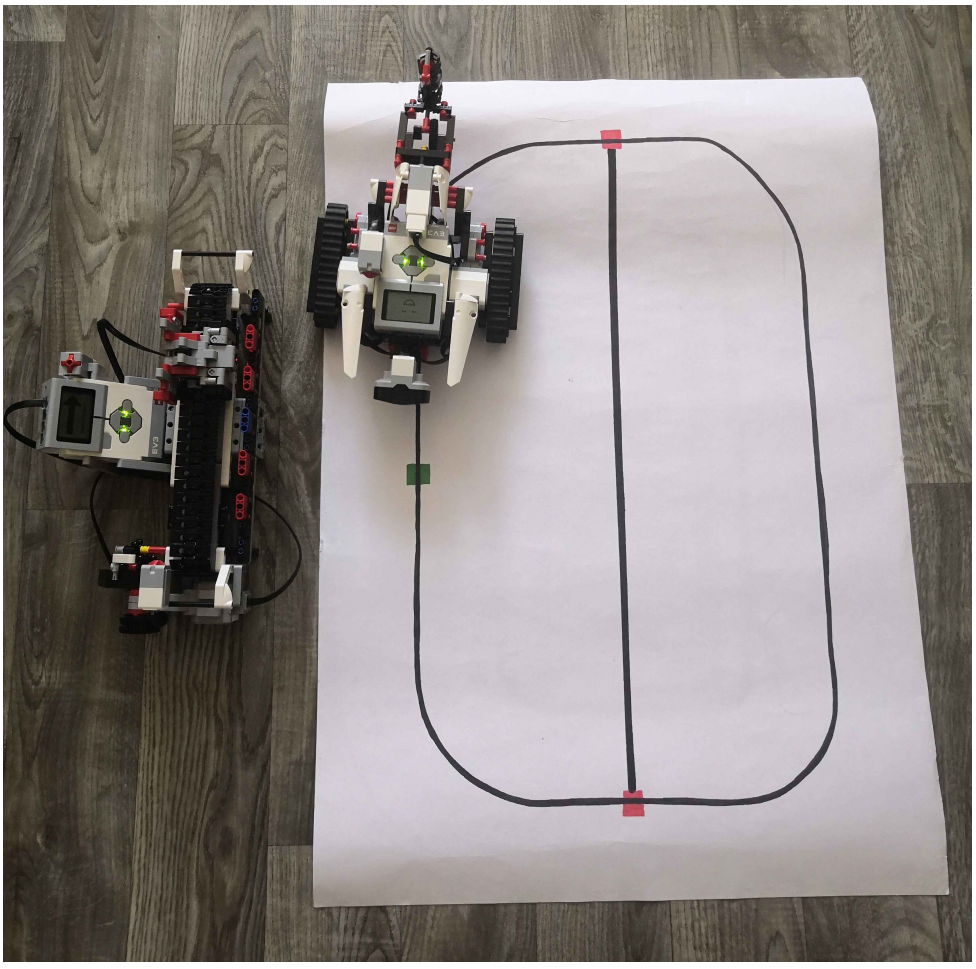
V této části provedeme celkové shrnutí funkcionality a na fotografiích ukážeme "typický průběh" experimentu.

Nejprve je potřeba mezi oběma roboty navázat Bluetooth spojení. Poté můžeme na jednotlivých robotech spustit nahrané programy.

Obrázek 19: Navázání Bluetooth spojení

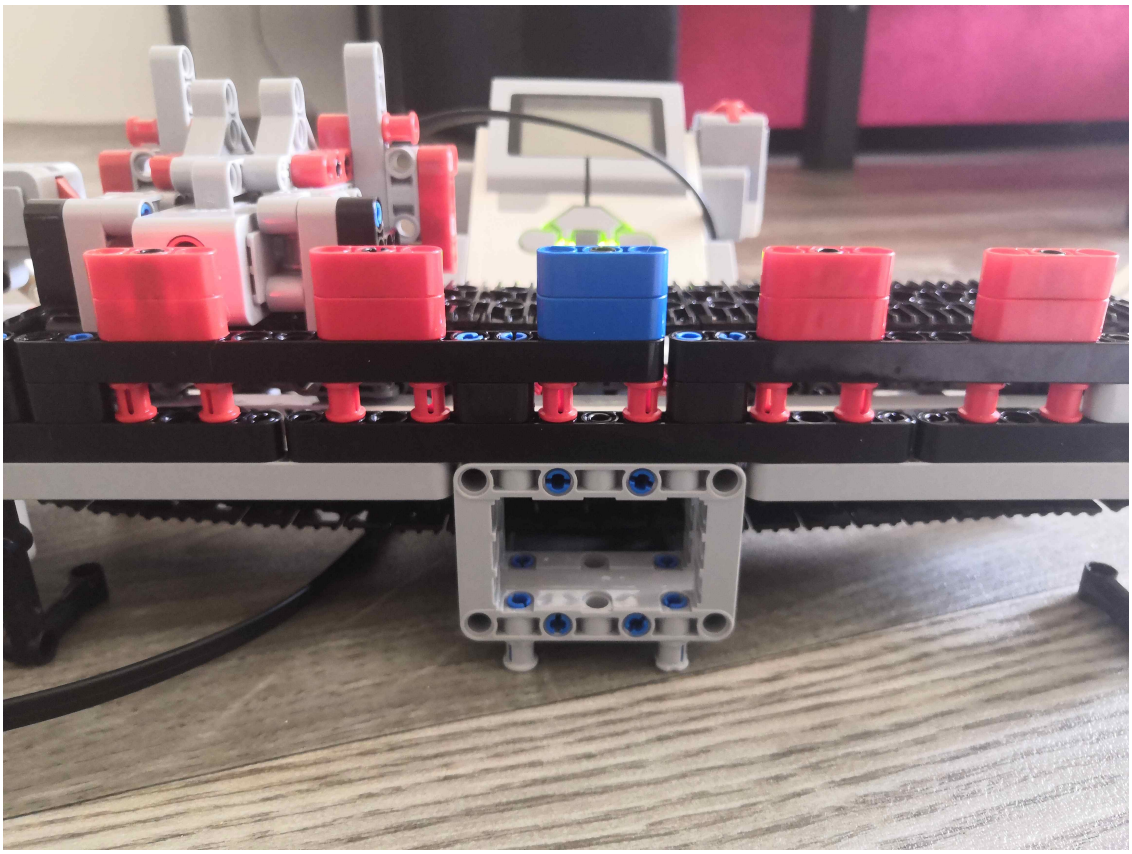
Obrázek 20: Spuštění programu

Po spuštění umístíme modul EV3-2 na dráhu tak, aby nejprve zatáčel směrem doprava.



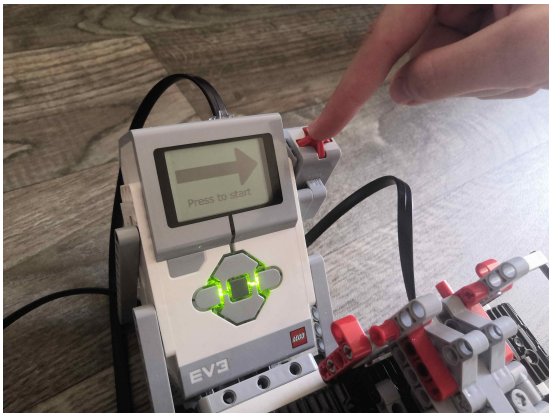
Obrázek 21: Umístění modulu EV3-2 na dráhu

Mezitím, co modul EV3-2 následuje dráhu a čeká na zprávu, vyskládáme dle potřeby barevné Lego kostky na modul EV3-1.



Obrázek 22: Vyskládaná zpráva

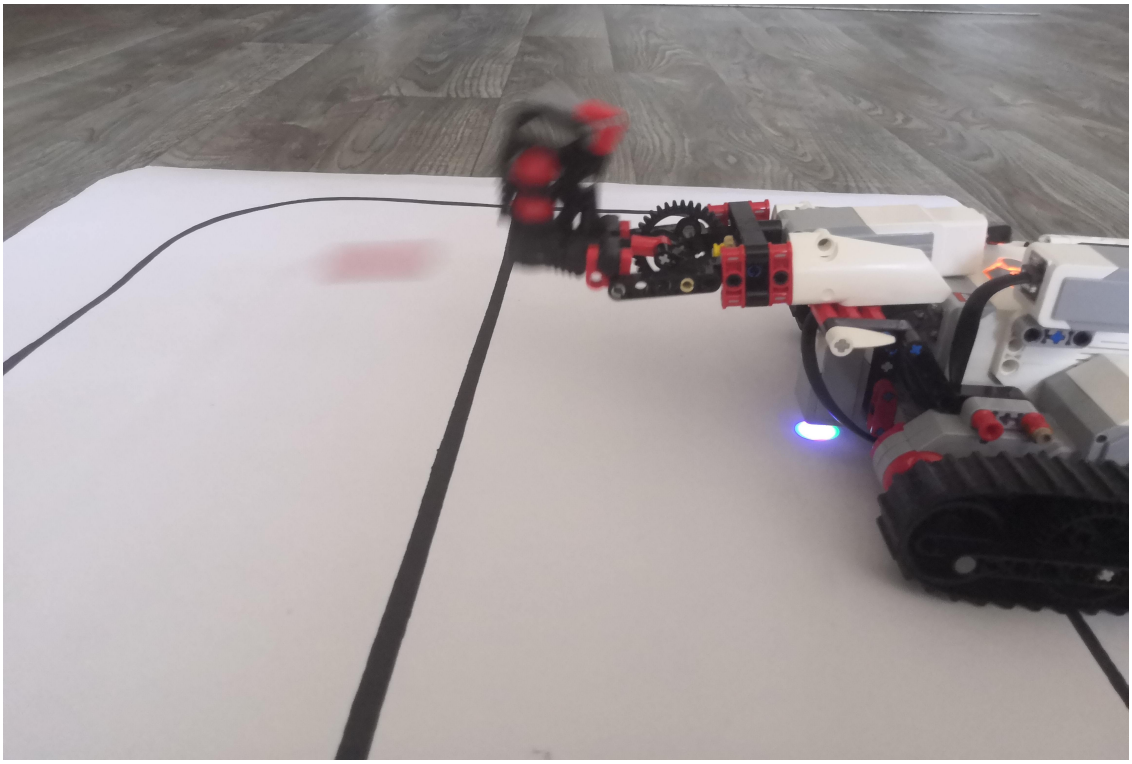
Následně stiskem inicializačního tlačítka přesuneme čtečku na začátek pásu. Dále zvolíme, zda chceme zapnout opravování chyb či nikoliv a opětovným stisknutím inicializačního tlačítka zahájíme čtení a odeslání zprávy.



Obrázek 23: Inicializační tlačítko

Obrázek 24: Volba zapnutí opravování chyb

Jakmile modul EV3-2 obdrží zprávu, provede odpovídající pokyn.



Obrázek 25: Modul EV3-2 provádí střelbu

Po provedení pokynu modul EV3-2 pokračuje v pohybu po dráze, na které se zrovna nachází, a vyčkává na další zprávu.

5.7 Alternativní uspořádání experimentu

V této podkapitole rozebereme možnosti, jak experiment upravit a jaké kódy zvolit, pokud budeme mít na experiment jiné požadavky. Jedná se spíše o diskuzi než o konkrétní návrhy dalších experimentů.

V našem experimentu jsme si zvolili, že chceme být schopni odesílat alespoň 4 pokyny a mít možnost opravit alespoň 1 chybu. Dále jsme se omezili pouze na binární kódy. V takovém případě jsme nuceni mít kódová slova o minimální délce 5, kratší být nemohou. Pokud bychom chtěli za každou cenu zkracovat, museli bychom zmírnit požadavky. Mírnit požadavek na opravování chyb nemá smysl, experiment má za úkol demonstrovat opravování chyb a tak by měl být schopen skutečně chyby opravovat. Pokud by nám stačil pouze jediný pokyn, opravíme takový počet chyb, jaká je jeho délka (cokoliv co není námi zvolené kódové slovo je špatně). Zde se však nejedná o demonstraci ani matematiky, ani základního principu samoopravných kódů. Smysl mají alespoň 2 kódová slova. V takovém případě lze zvolit opakovací kód $C = \{000, 111\}$.

Opakovací kódy jsou obecně dobrá volba pro demonstraci základního principu opravování chyb. Dekódování nepředpokládá prakticky žádnou znalost matematiky a kód můžeme libovolně prodlužovat a tím navyšovat počet opravitelných chyb. Nevýhodou je limit na právě 2 kódová slova (pro binární kódy).

Pokud naopak trváme na větším počtu pokynů, musíme pochopitelně kódová slova prodlužovat. Když bychom zůstali u délky 5, jsme schopni sestrojit lineární $[5, 3, 2]$-kód

$$C = \{00000, 11100, 10010, 01001, 01110, 10101, 11011, 00111\}$$

s osmi kódovými slovy a generující maticí

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

Tento kód je mimochodem duálním kódem k $[5, 2, 3]$-kódu z experimentu. Navýšení počtu pokynů mělo ovšem za následek pokles minimální vzdálenosti a ztrátu možnosti opravovat jednu libovolnou chybu. Pro 8 kódových slov a minimální vzdálenost 3 (oprava 1 libovolné chyby) musíme prodloužit alespoň na délku 6, čímž získáme $[6, 3, 3]$-kód

$$C = \{000000, 100110, 010011, 001101, 110101, 011110, 101011, 111000\}$$

s generující maticí

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Co se týče navyšování počtu opravitelných chyb, pro 2 chyby potřebujeme kód s minimální vzdáleností alespoň 5. Že binární kód s minimální vzdáleností 5 a délkou 5 může mít nejvýše 2 kódová slova je jasné (opět opakovací kód). Více než 2 slova ovšem nemůže mít ani kód délky 6 (zjistíme např. z Hammingova odhadu), ani kód délky 7 (viz [3]). Pokud chceme kód s minimální vzdáleností 5 s více než dvěma kódovými slovy, musíme prodloužit až na délku 8, čímž dostaneme $[8, 2, 5]$-kód

$$C = \{00000000, 10110111, 01001111, 11111000\}$$

s generující maticí

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

V našem experimentu, jak už bylo rozebráno dříve, jsme si ze zjevných důvodů nedovolili použít Reed-Solomonovy kódy. Pokud by ovšem nějaký jiný alternativní experiment jejich použití

umožňoval, museli bychom zároveň také přejít k větším abecedám, protože postupem popsaným v bakalářské práci [13] jsme schopni zkonstruovat jediný binární RS-kód $C = \{0, 1\}$. Pokud bychom chtěli opravit alespoň jednu chybu, museli bychom jako abecedu chápat konečné těleso $GF(4)$, nad kterým jsme schopni zkonstruovat RS $[3, 1]$-kód

$$C = \{000, 111, \alpha\alpha\alpha, \alpha^2\alpha^2\alpha^2\}.$$

Můžeme si všimnout, že se v podstatě nejedná o nic jiného než o kvaternární opakovací kód.

Výhodou oproti binárnímu kódu je, že můžeme opravit 1 chybu a přitom mít 4 kódová slova délky pouze 3. Nevýhodou je, že potřebujeme 4 symboly. Pokud bychom tedy chtěli kódová slova přenášet binárním kanálem (což v praxi musíme), museli bychom symboly stejně reprezentovat bitovou sekvencí. Symbol 0 můžeme reprezentovat jako 00, symbol 1 jako 01, symbol α jako 10 a symbol α^2 jako 11. Bitová reprezentace RS $[3, 1]$-kódu by tudíž byla

$$C = \{000000, 010101, 101010, 111111\},$$

čímž bychom stejně ztratili výhodu zkrácené délky, protože bychom potřebovali 6 bitů na každé kódové slovo. Pro malé kódy určené pro binární komunikační kanály se proto zjevně vyplatí situaci zbytečně nekomplikovat a zůstat u obecných binárních lineárních blokových kódů.

Pokud by ovšem byl komunikační kanál realizován stejně, jako je tomu u našeho experimentu, výhodu zkrácené délky uplatnit můžeme, protože jednoduše namísto dvou barevných lego kostek použijeme čtyři. Kdybychom ale použili jiný RS-kód, který by nebyl pouhým opakovacím kódem, museli bychom realizovat dekódovací algoritmy, které jsou matematicky velice složité a v prostředí vizuálního programovacího jazyka pro Lego Mindstorms nerealizovatelné.

Nyní se budeme zabývat nikoliv alternativními samoopravnými kódy, které můžeme použít, ale alternativní realizací samotných robotů.

Co se týče Modulu EV3-2 (robot reagující na pokyny), je prostor pro alternativní realizaci obrovský. Můžeme ubrat počet pokynů, přidat počet pokynů nebo pokyny úplně změnit na jiné. Stavebnice Lego Mindstorms v tomto ohledu umožňuje absolutní volnost. Důležitým aspektem experimentu je hlavně demonstrování opravování chyb, jaké robot vykonává konkrétní pokyny není tolik důležité, dokud experiment přitahuje pozornost.

Alternativní realizace Modulu EV3-1 (čtečky) je už poměrně složitější. Že došlo k chybě musí být zjevné a chyby nejlépe musejí být plně pod kontrolou uživatele. Například by bylo možné použít barevný senzor nikoliv pro čtení barev, ale jako senzor jasu a realizovat reprezentaci zprávy na bázi "vyblikávání" (princip Morseova kódu). V takovém případě by ovšem bylo potřeba mít

programovatelnou lampičku, která by na základě uživatelského vstupu blikala odpovídajícím způsobem. Pokus na bázi jasu by dále byl velmi závislý na osvětlení prostředí. Byl by problém nastavit hodnotu prahu, tzn. určit, kolik procent jasu už interpretujeme jako bliknutí. Hodnota by se musela neustále upravovat, protože v tmavé místnosti by byl práh jiný než v přesvětlené.

Další možností je samozřejmě také vůbec nepoužít stavebnici Lego Mindstorms. Případných náhrad určitě existuje celá řada, stavebnice Mindstorms ovšem plně splnila veškeré požadavky a v našem případě nebylo potřeba žádnou alternativu vyhledávat.

Abychom problematiku návrhu experimentu a volby kódu shrnuli, pro danou situaci a námi předem stanovené požadavky byl zvolen nejoptimálnější kód. Jeho kódování a dekódování není složité, princip fungování samoopravných kódů lze snadno demonstrovat a pro binární kód se čtyřmi pokyny a jednou opravitelnou chybou je nejkratší možný. Při volbě většího kódu je možno buď mít větší abecedu, pak bychom ovšem museli přejít ke složitějším a méně snadno uchopitelným třídám kódů, nebo bychom museli kód prodlužovat. V úvahu by potom přicházel binární [6,3,3]-kód s osmi kódovými slovy. Pokud bychom trvali na navýšení počtu opravitelných chyb na dvě, zvolit můžeme binární [8,2,5]-kód se čtyřmi kódovými slovy. Prodloužení kódu by zároveň znamenalo nutnost přestavby robota interpretujícího zprávu. Bylo by potřeba celý čtecí pás rozšířit, abychom mohli vyskládat větší počet Lego kostiček. Rozšiřování ovšem zbytečně snižuje přehlednost celého experimentu. Případně je možno zvolit úplně jiný princip čtení, námi zvolený postup se však ukázal jako plně vyhovující.

6 Závěr

Cílem této diplomové práce byl návrh a následná realizace experimentu, který představí princip fungování samoopravných kódů laické veřejnosti, čímž popularizuje matematiku. Diplomová práce navazuje na předchozí bakalářskou práci [13].

Nejprve bylo potřeba definovat nezbytné pojmy a uvést matematický aparát, aby bylo možno navrhnout a hlavně srovnat různé samoopravné kódy. Velká část kapitoly byla věnována odhadům maximálního počtu kódových slov pro kód s danými parametry. Kromě uvedení samotných odhadů s jejich důkazy jsme se například zabývali i jejich srovnáním.

Z důvodu relativní jednoduchosti kódování a dekódování jsme následně přešli ke konkrétní třídě samoopravných kódů, lineárním kódům. Kromě potřebných definic a vět jsme se zabývali také výpočty spolehlivosti přenosu a obtížností konstrukce.

Dalším tématem pak byla konkrétní reálná a hlavně aktuální aplikace, která samoopravné kódy využívá, americká vesmírná sonda Juno. Předně nás zajímalo, s jakými konkrétními samoopravnými kódy se zde můžeme setkat a za jakým účelem se používají. Juno pak posloužila jako vzor pro experiment.

Následně jsme detailně rozebrali kód, jež byl nakonec v experimentu použit. Uvedli jsme veškeré jeho parametry, vypočetli jeho spolehlivost a poskytli všechny náležitosti k jeho používání.

Nakonec jsme přešli k hlavní části této práce, k samotnému experimentu. Ten jsme navrhli s využitím stavebnice Lego Mindstorms, kterou jsme představili a uvedli také možnosti jejího programování. Navrženi byli roboti a způsob komunikace mezi nimi tak, aby byly zřetelně demonstrovány principy fungování samoopravných kódů. Vše, včetně zdrojových kódů, je v práci podrobně zdokumentováno. V neposlední řadě jsme shrnuli přednosti i úskalí zvolených postupů.

Použitá stavebnice zůstane k dalšímu využití k dispozici na katedře aplikované matematiky. Původní realizaci je proto možno dále rozšiřovat a rozvíjet. Diskuzi k možným alternativním uspořádáním je věnována poslední kapitola 5.7.

Hlavní přínos práce tkví v oblasti propagace a popularizace matematiky. Experiment se objevil na propagačních akcích, pořádaných univerzitou, konkrétně 21.11.2019 na Zlepši si techniku s VŠB-TUO a 24.01.2020 na dni otevřených dveří FEI VŠB-TUO. K těmto akcím vznikl také doprovodný materiál (letáky), který je v příloze B. Roboty lze taktéž použít jako učební pomůcku v základních kurzech abstraktní algebry nebo teorie kódování.

Při návrhu a programování jsem si jako autor měl možnost projít procesem konstrukce a implementace samoopravných kódů, čelit problémům s tím spojeným a tím si utřídit teoretické znalosti nabyté již při psaní bakalářské práce. Zároveň jsem díky diplomové práci získal možnost zúčastnit se propagačních akcí, kde kromě zlepšení komunikačních dovedností oceňuji hlavně obdržení okamžité zpětné vazby od návštěvníků.

Zvolená realizace se ukázala jako vhodná, díky zábavné formě byli i žáci prvního stupně základních škol schopni pochopit, jak s roboty zacházet a na jakém principu funguje opravování chyb. Zároveň experiment není až příliš jednoduchý a ocenili jej i pokročilejší studenti a dospělí z řad návštěvníků. I přes problémy nastíněné v kapitole 5.5 se nakonec podařilo dostát původní vizi a splněny byly veškeré předem vytyčené požadavky.

Reference

[1] GALLIAN, Joseph, A. *Contemporary abstract algebra*. 5th ed. Boston: Houghton Mifflin, 2002. ISBN 978-0618122141.

[2] REED, Irving, S. - CHEN, Xuemin. *Error-control coding for data networks*. Boston: Kluwer Academic Publishers, 1999. ISBN 0-7923-8528-4.

[3] HILL, Raymond. *A first course in coding theory*. New York: Oxford University Press, 1986. ISBN 0-19-853803-0.

[4] HANKERSON, Darrel. - HOFFMAN, Gary. *Coding Theory and Cryptography: The Essentials*. Second edition. New York: CRC Press, 2000. ISBN 0-8247-0465-7.

[5] ROMAN, Steven. *Introduction to coding and information theory*. New York: Springer, 1997. ISBN 0387947043.

[6] VERMANI, Lekh. *Elements of Algebraic Coding Theory*. New York: Springer, 1996. ISBN 9780412573804.

[7] MUKAI, Ryan. - HANSEN, David. - MITTSKUS, Anthony. - TAYLOR, Jim. - DANOS, Monika. *Juno Telecommunication*. DESCANSO Design and Performance Summary Series [online]. 2012, (Article 16) [cit. 2020-02-13]. Dostupné z: `https://descanso.jpl.nasa.gov/DPSummary/Juno_DESCANSO_Post121106H--Compact.pdf`.

[8] CHEN, Wai-Kai. *The electrical engineering handbook*. Boston: Elsevier Academic Press, 2005. ISBN 978-0-12-170960-0.

[9] *Why the only binary MDS codes are trivial ones?* Mathematics Stack Exchange [online]. [cit. 2020-02-18]. Dostupné z: `https://math.stackexchange.com/questions/1066780/why-the-only-binary-mds-codes-are-trivial-ones`.

[10] *CCSDS recommended standard for TM synchronization and channel coding.*[online]. Washington, DC, USA: CCSDS. 2017 [cit. 2020-02-18]. Dostupné z: `https://public.ccsds.org/Pubs/131x0b3e1.pdf`.

[11] *Juno - Mission to Jupiter*. NASA [online]. 2011 [cit. 18.02.2020]. Dostupné z: `https://www.nasa.gov/mission_pages/juno/main/index.html`.

[12] BERROU, Claude. - GLAVIEUX, Alain. - THITIMAJSHIMA, Punya. *Near Shannon limit error-correcting coding and decoding: Turbo-codes*. Proc. ICC'93, Geneva, Switzerland. 1064 - 1070 vol.2. 10.1109/ICC.1993.397441, 1993.

[13] SALAMON, Jakub. *Samoopravné kódy a jejich aplikace*. Ostrava, 2018. Bakalářská práce. VŠB-TUO. Vedoucí práce Mgr. Tereza Kovářová, Ph.D.

[14] *LEGO MINDSTORMS EV3 Help*. [online]. [cit. 2020-03-03]. Dostupné z: https://ev3-help-online.api.education.lego.com/Education/en-gb/.

[15] *LEGO MINDSTORMS ke stažení*. [online]. [cit. 2020-03-03]. Dostupné z: https://www.lego.com/cs-cz/themes/mindstorms/downloads.

[16] *LEGO MINDSTORMS EV3 Support from MATLAB*. [online]. [cit. 2020-03-03]. Dostupné z: https://www.mathworks.com/hardware-support/lego-mindstorms-ev3-matlab.html.

[17] *ROBOTC.net*. [online]. [cit. 2020-03-03]. Dostupné z: http://www.robotc.net/.

[18] *ROBOTC for LEGO Mindstorms 4.X - Users Manual*. Robomatter Inc. [online]. 2016 [cit. 2020-03-03]. Dostupné z: http://help.robotc.net/WebHelpMindstorms/index.htm.

[19] NASA : Commons.wikimedia.org: File:JUNO - PIA13746.jpg [online]. 3 January 2011 [cit. 2020-03-03]. Dostupný pod licencí Creative Commons na www: https://commons.wikimedia.org/wiki/File:JUNO_-_PIA13746.jpg.

A Zdrojový kód pro vysílač napsaný v RobotC

Uvedený zdrojový kód pro vysílač (Modul EV-1), napsaný v jazyce RobotC, je ekvivalent zdrojového kódu skutečně použitého v experimentu s tím rozdílem, že chybí odesílání zpracované zprávy do druhého robota, protože jazyk RobotC neumožňuje Bluetooth komunikaci mezi dvěma zařízeními.

```
#pragma config(Sensor, S1,  beltButton,    sensorEV3_Touch)
#pragma config(Sensor, S2,  modeButton,    sensorEV3_Touch)
#pragma config(Sensor, S3,  color,         sensorEV3_Color)
#pragma config(Motor, motorD, beltMotor, tmotorEV3_Large, PIDControl, encoder)

//vykresli sipku na display
void drawRightArrow(const int left, const int top, const int right, const int
    bottom)
{
   int arrowTop = top+(top-bottom)/3;
   int arrowBottom = bottom-(top-bottom)/3;
   fillRect(left,top,right,bottom);
   for(int i = 0; i <= (arrowTop-arrowBottom)/2;i++)
   {
      drawLine(right+i,arrowTop-i,right+i,arrowBottom+i);
   }
}

//volba zapnuti nebo vypnuti opravovani chyb
bool setEC()
{
   setLEDColor(ledGreen);
   eraseDisplay();
   displayBigStringAt(0,100, "Turn on error");
   displayBigStringAt(0,80, "correction?");
   displayBigStringAt(0,20, "YES");
   displayBigStringAt(150,20, "NO");

   while(true)
   {
      if(getButtonPress(buttonLeft) == 1)
      {
```

```
            return true;
        }

        if(getButtonPress(buttonRight) == 1)
        {
            return false;
        }
    }
}

//presune barevny senzor na zacatek pasu
void initialize()
{
    setLEDColor(ledOrange);
    eraseDisplay();

    while(getTouchValue(beltButton)==0)
    {
        setMotor(beltMotor,-30);
    }
    setMotor(beltMotor,10);
    sleep(500);
    stopAllMotors();
    getColorName(color);
    setLEDColor(ledGreen);
}
//vraci hodnotu na zaklade snimane barvy
void readColor(string& bit)
{
    switch (getColorName(color))
    {
    case colorBlue: bit = "0"; break;
    case colorGreen: bit = "0"; break;
    case colorYellow: bit = "1"; break;
    case colorRed: bit = "1"; break;
    default: bit = 'X'; break;
    }
}
```

```
//posune senzor a nasnima barvu
void readOne(string& message, string& bit, int speed, int sleepTime1,int
    sleepTime2)
{
   setMotor(beltMotor,speed);
   sleep(sleepTime1);
   stopAllMotors();
   readColor(bit);
   strcat(message,bit);
   sleep(sleepTime2);
}

//nasnima celou zpravu
void readMessage(string& message)
{
   string bit = "";
   setLEDColor(ledOrange);
   eraseDisplay();
   displayBigStringAt(0,100, "Reading...");
   readOne(message,bit,0,0,100);
   readOne(message,bit,20,650,500);
   readOne(message,bit,20,650,500);
   readOne(message,bit,19,650,500);
   readOne(message,bit,19,650,500);
   eraseDisplay();
   setLEDColor(ledGreen);
}

//nahrazuje symboly X za 0
void xReplace(string& str)
{
   char tmp[6];
   for(int i = 0; i < 5; i++)
   {
      tmp[i] = stringGetChar(str,i);
      if (tmp[i] == 'X')
         tmp[i] = '0';
   }
   tmp[5] = (char)0;
```

```
   stringFromChars(str,tmp);
}

//opravovani zpravy na zaklade standard array dekodovani
bool correctMessage(string message, string& corrected)
{
   const int cols = 4;
   const int rows = 8;
   string standardArray[cols][rows] =
   {{"00000","00001","00010","00100","01000","10000","11000","10001"},
      {"01101","01100","01111","01001","00101","11101","10101","11100"},
      {"10110","10111","10100","10010","11110","00110","01110","00111"},
      {"11011","11010","11001","11111","10011","01011","00011","01010"}};

   for (int i = 0; i < cols; i++)
   {
      for (int j = 0; j < rows; j++)
      {
         if (standardArray[i][j] == message)
         {
            corrected = standardArray[i][0];
            if (j != 0)
            {
               return true;
            }
            else return false;
         }
      }
   }

   string messageCopy = message;
   xReplace(messageCopy);
   correctMessage(messageCopy,corrected);
   return true;
}
```

```
task main()
{
   bool initialized = false;
   setLEDColor(ledGreen);
   resetBumpedValue(modeButton);
   string message = "";
   string correctedMessage = "";
   bool EC = false;
   bool isError = false;

   while(true)
   {
      eraseDisplay();
      message = "";
      correctedMessage = "";
      EC = false;
      isError = false;

      while(initialized == false)
      {
         setLEDColor(ledGreen);
         drawRightArrow(10,100,140,60);
         displayBigStringAt(10, 35, "Press to start");

         if (getBumpedValue(modeButton) != 0)
         {
            initialize();
            EC = setEC();

            eraseDisplay();
            displayBigStringAt(0,100, "Ready!");
            displayBigStringAt(0,60, "Press again to");
            displayBigStringAt(0,40, "send message");
            drawRightArrow(80,103,150,83);

            initialized = true;
            resetBumpedValue(modeButton);
         }
      }
```

```
      while(initialized == true)
      {
         if (getBumpedValue(modeButton) != 0)
         {
            readMessage(message);
            if (EC)
            {
               isError = correctMessage(message,correctedMessage);
            }
            if (isError)
            {
               setLEDColor(ledRedFlash);
               displayBigStringAt(0,120, "Error detected!");
               displayBigStringAt(0,100, "Message:");
               displayBigStringAt(0,80, message);
               displayBigStringAt(0,60, "Changed to:");
               displayBigStringAt(0,40, correctedMessage);
               displayBigStringAt(0,20, "Sending...");
            }
            else
            {
               setLEDColor(ledGreenFlash);
               displayBigStringAt(0,100, "Sending");
               displayBigStringAt(0,80, "message:");
               displayBigStringAt(0,40, message);
            }
            sleep(3000);
            initialized = false;
            resetBumpedValue(modeButton);
         }
      }
   }
}
```

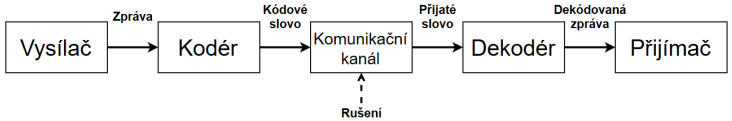
Samoopravné kódy

Co jsou samoopravné kódy?

Chceme-li přenášet libovolnou informaci z bodu A (vysílač) do bodu B (přijímač), většinou se nevyhneme rušení, které danou informaci pozmění či jinak znečitelní. A protože si často nemůžeme dovolit opakovaný přenos pro kontrolu (např. příliš dlouho trvá nebo stojí moc pěněz), hodilo by se nám mít jistotu, že chybnou informaci stejně dokážeme správně interpretovat. Za tímto účelem se v praxi využívají samoopravné kódy.

Základní princip

Představme si, že chceme z vysílače odeslat do přijímače jednu ze dvou možných zpráv. "Ano" - reprezentované jedničkou a "Ne" - reprezentované nulou. Pokud odešleme například zprávu 0 a po cestě vlivem rušení dojde ke změně zprávy (0 se zamění za 1), přijímač nemá šanci poznat, že došlo k chybě a zpráva, kterou obdržel, není správná.
Pokud ovšem obě zprávy uměle prodloužíme o zdánlivě zbytečné (redundantní) symboly (namísto 0 budeme odesílat 000, případně místo 1 odešleme 111), získáme tím schopnost opravit libovolnou jednu nastalou chybu. Přijímač totiž ví, že pokud přijme cokoliv jiného než 000 nebo 111 (například 001), došlo k chybě. Zároveň je možné si domyslet, že 001 má nejblíže k 000, a tudíž původní zpráva byla nejspíš (ale ne jistě) 000.
Tento jednoduchý samoopravný kód nese označení binární opakovací kód.



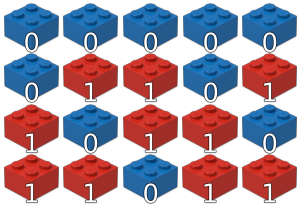
Kde je najdeme?

Samoopravné kódy jako takové dnes nacházejí své uplatnění prakticky všude, kde se nějakým způsobem přenášejí data. Každý kód se podle svých vlastností hodí do jiných situací.
Setkat se s nimi můžeme nejen u zařízení pro ukládání dat, jako jsou CD, DVD a Blu-Ray disky, ale také u čárových kódů (QR-kódů), bezdrátových sítí, digitální televize a satelitní komunikace, včetně přenosu dat z vesmíru.

Vyzkoušejte si...

princip samoopravných kódů na připravených LEGO robotech. Využijte barevné kostičky k odeslání jednoho ze čtyř pokynů. Vyskládejte správnou kombinaci nebo kombinaci s jednou chybou. Libovolnou kostičku buď vyměňte za jinou, nebo zakryjte. Při zapnutém opravování chyb robot i tak provede správný pokyn. V tabulce Standard array jsou uvedeny všechny možné kombinace, které lze vyskládat. Všechny prvky ze stejného sloupce tabulky se opraví a interpretují jako pokyn v prvním řádku.

Seznam pokynů



- Změna směru
- Přechod na vnější dráhu
- Přechod na vnitřní dráhu
- Střelba

Standard array

00000	**01101**	**10110**	**11011**	
00001	01100	10111	11010	
00010	01111	10100	11001	
00100	01001	10010	11111	1 chyba
01000	00101	11110	10011	
10000	11101	00110	01011	
11000	10101	01110	00011	2 chyby
10001	11100	00111	01010	

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA APLIKOVANÉ MATEMATIKY
AM.VSB.CZ

Kódování a dekódování

Jaký typ kódu byl použit?

V našem příkladě s roboty byl použit kód C spadající do skupiny tzv. blokových kódů, jelikož všechna kódová slova mají stejnou délku. Trochu konkrétněji se pak jedná o tzv. lineární kód. Ke kódování a dekódování lze využít aparát lineární algebry. K jednotlivým kódovým slovům se přistupuje jako k vektorům.

Vlastnosti kódu

Použitý kód C se v řeči samoopravných kódů označuje jako binární lineární $[5, 2]$-kód. Uvedené číslovky znamenají, že kód je schopen zakódovat zprávy o délce 2 přidáním redundance, čímž jej prodloužíme na celkovou délku 5.
Zprávy, které je tento kód schopen zakódovat, jsou čtyři, konkrétně 00,01,10 a 11. Po přidání redundance dostáváme čtyři kódová slova 00000, 01101, 10110 a 11011.
Počet chyb, které kód opraví, se odvíjí od tzv. minimální vzdálenosti kódu d. Pro náš kód platí $d = 3$, je proto schopen opravit libovolnou jednu chybu a zaregistrovat dvě.

Kódování

Ke kódování lineárních kódů se využivá tzv. generující matice G. Zprávu, kterou chceme zakódovat, vynásobíme zprava maticí G.

Vybereme tedy například zprávu 01 a zakódujeme ji:

$$\vec{z} \cdot G = \vec{c}$$

$$\begin{bmatrix} 0\,1 \end{bmatrix} \cdot \begin{bmatrix} 0\,1\,1\,0\,1 \\ 1\,0\,1\,1\,0 \end{bmatrix} = \begin{bmatrix} 1\,0\,1\,1\,0 \end{bmatrix}.$$

Řádky matice G jsou bázové vektory podprostoru C (kódu) ve vektorovém prostoru $V(5, 2)$ nad konečným tělesem $GF(2)$.

Dekódování

Dekódování bývá obecně mnohem těžší než kódování. Jeho součástí je i samotné opravování chyb. U lineárních kódů máme na výběr ze dvou možných způsobů dekódování.

První z nich využívá vyhledávácí tabulku "Standard array" (viz list s pokyny k ovládání robota). Na tomto způsobu je složitá hlavně konstrukce samotné tabulky. Pak už v ní jen jednoduše nalezneme přijatou zprávu a podíváme se na první řádek ve stejném sloupci. Tento dekódovací algoritmus je jednoduchý a snadno stravitelný, v praxi se ovšem příliš nepoužívá, jelikož pro velké kódy (běžně se používají kódy o délce 256 bitů) zabírá Standard array příliš paměti.

Paměťovou náročnost do jisté míry řeší druhý dekódovací algoritmus, "syndromové dekódování". Využívá se existence duálního kódu, jehož generující matice H je ortogonální k matici G, tj. $G \cdot H^T = \mathbf{0}$.

Pro námi použitý kód vypadá matice H a syndromová tabulka následovně:

$$H = \begin{bmatrix} 1\,1\,1\,0\,0 \\ 1\,0\,0\,1\,0 \\ 0\,1\,0\,0\,1 \end{bmatrix}$$

$f(\vec{y})$	$S(\vec{y})$
00000	000
00001	001
00010	010
00100	100
01000	101
10000	110
11000	011
10001	111

, kde $S(\vec{y}) = \vec{y} \cdot H^T$

Dekódování probíhá tak, že pro přijaté slovo $\vec{y}$ nejprve vypočteme syndrom $S(\vec{y})$, nalezneme jej v tabulce, zůstaneme na stejném řádku a podíváme se na sloupec $f(\vec{y})$. Následně provedeme výpočet $\vec{c} = \vec{y} - f(\vec{y})$, čímž provedeme dekódování na kódové slovo $\vec{c}$.

Reálná aplikace samoopravných kódů

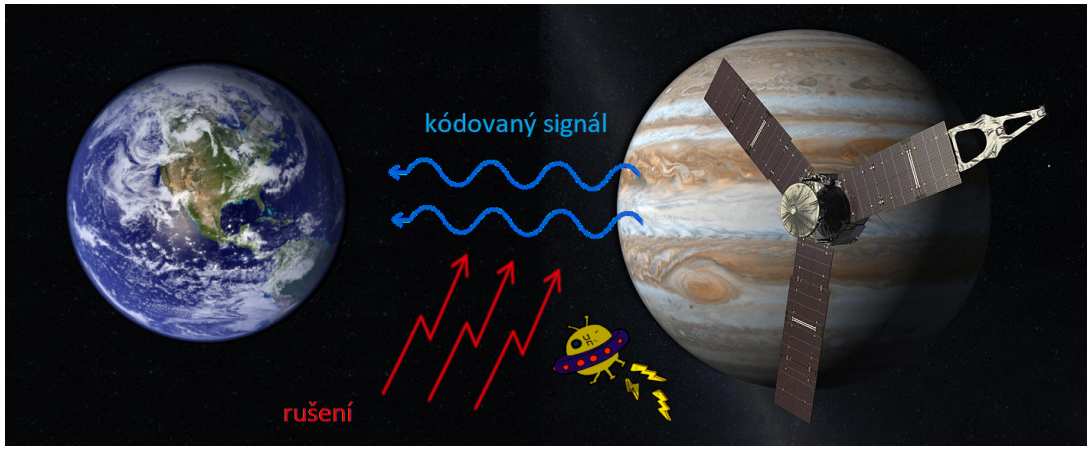
Sonda Juno - Mise Jupiter

Start: srpen 2011 **Přílet k Jupiteru**: červenec 2016 **Plánovaný návrat**: červenec 2021

Americká kosmická sonda **Juno** v rámci programu 'New Frontiers' doletěla 5. července 2016 k planetě **Jupiter**. Odstartovala v pátek 5. srpna 2011 z kosmodromu na mysu Canaveral na Floridě pomocí rakety Atlas V. Využila manévru „gravitačního praku“. Nejprve zamířila k Marsu a v říjnu 2013 proletěla kolem Země ve vzdálenosti 564 km, přičemž získala rychlost a energii k letu k Jupiteru. V červenci 2016 byla sonda navedena na polární oběžnou dráhu Jupiteru odkud zkoumá jeho atmosféru a měsíce. Juno vysílá k Zemi mnoho vědecky zajímavých informací a také fascinující snímky i videa planety Jupiter. `https://cs.wikipedia.org/wiki/Juno_(sonda)`

Země

Juno a Jupiter



Jaké kódy jsou použity ke komunikaci mezi Juno a Zemí?

Ke komunikaci sonda používá kombinaci **zřetězených konvolučních a Reed-Solomonových** nebo **turbo** samoopravných kódů s informačním poměrem 1/6. Většina rušení přenosu pochází z prostorové plazmy mezi Jupiterem a Zemí nebo ze zemské atmosféry.

Více informací najdete na webu
`https://space.stackexchange.com/questions/2346/resilience-to-data-transmission-errors-of-the-juno-spacecraft`

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA APLIKOVANÉ MATEMATIKY
AM.VSB.CZ